DOCUMENT RESUME

ED 355 033 PS 021 220

AUTHOR	Halford, Graeme S.; And Others
TITLE	Acquisition of Reasoning: A Computational Model of Strategy Development in Transitive Inference.
INSTITUTION	Queensland Univ., Brisbane (Australia). Centre for Human Information Processing and Problem Solving.
REPORT NO	CHIPPS-TR-92-1
PUB DATE	92
NOTE	101p.; This work was supported by a grant from the Australian Research Grants Scheme.
PUB TYPE	Reports - Research/Technical (143)
EDRS PRICE	MF01/PC05 Plus Postage.
DESCRIPTORS	*Child Development; Children; *Cognitive Development; *Cognitive Mapping; *Computer Simulation; Concept Formation; Critical Thinking; Foreign Countries; Models; Problem Solving; Schemata (Cognition)
IDENTIFIERS	*Analogical Reasoning; Ordering Operations; PRISM Programing Language; *Transitive Inferences

ABSTRACT

This paper describes a computer-simulation model of the way in which basic reasoning processes develop in children. The model, based on PRISM-II programming language, was designed to reflect the manner in which world knowledge can be used to construct strategies for reasoning. The model learns strategies for performing transitive inference by using knowledge that is acquired through play experiences with sets of objects. Five experiments testing predictions from the model are reported. Experiment 1 tested children's ability to recognize ordered sets, an ability required for the development of transitive inference strategies. Experiments 2 and 3 tested the prediction that children would recognize when the order of a set of objects is indeterminate and found that the ability to make this recognition was present from the age of 7 years. Experiments 4 and 5 assessed strategies children used in a varied set of transitive inference problems, and found no qualitative differences in strategies as a function of age. Six appendices making up almost half the document include the set of productions existing before the model begins operation and lists of productions produced during various problem forms. (Contains 75 references.) (MM)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☒ This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

Centre for Human Information Processing and Problem Solving

Department of Psychology
The University of Queensland
Queensland, Australia 4072

ED355033

Acquisition of reasoning:

A computational model of strategy development in transitive inference

Graeme S. Halford, Murray T. Maybery, Susan B. Smith,
John D. Bain, J. Campbell Dickson, Mavis E. Kelly, and J.E.M. Stewart

PS021220

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
G.S. Halford
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

CHIPPS–TR–92–1

BEST COPY AVAILABLE

Acquisition of Reasoning:

A Computational Model of Strategy Development in Transitive Inference

Technical report 92/1

Centre for Human Information Processing and Problem Solving.

Graeme S. Halford, Murray T. Maybery, Susan B. Smith, John D. Bain,

J. Campbell Dickson, Mavis E. Kelly, and J.E.M. Stewart

University of Queensland

Running head: Acquisition of Reasoning

This work was supported by a grant from the Australian Research Grants Scheme.
Author's address: Graeme Halford, Psychology Department, University of Queensland, 4072, AUSTRALIA. Email: gsh@psych.psy.uq.oz.au. Fax: 61 7 365 4466.

Abstract

A computer-simulation model of strategy development in N-term series (transitive inference) problems is presented. The model learns strategies for performing N-term series using knowledge that can be acquired through play experiences with sets of objects. It is theorised that declarative knowledge acquired through play with sets of objects is utilized by analogical reasoning mechanisms to guide the development of strategies. Procedural knowledge, also acquired through play, is utilized by means-end analysis. These two types of knowledge are used to build strategies as they are required to solve transitive inference problems. Once developed, strategies are strengthened by associative learning mechanisms. Thus the model blends associative and metacognitive strategy development mechanisms. If strategies appropriate to a task are available they are applied, new strategies being developed as required. The model simulates the transition from no strategies, through primitive strategies in which premises are processed singly, leading to errors, to adequate strategies in which premises are integrated, leading to correct performance. The simulation is a self-modifying production system model, based on PRISM-II.

Five experiments testing predictions from the model are reported. Experiment 1 tests children's ability to recognize ordered sets, an ability that is at the core of the declarative knowledge required to develop transitive inference strategies. It is found that children of five years can discriminate ordered from unordered sets. Experiments 2 and 3 test the prediction that children should be able to recognize when the order of a set of objects is indeterminate. This ability is found to be present from the age of 7 years. Experiments 4 and 5 make detailed assessments of the strategies children employ in a varied set of transitive inference problems. No qualitative differences in strategies are found as a function of age. However younger children, as well as older children and adults when under high processing loads, make errors due to working memory failure. The model is interpreted as a sufficiency test for a theory of autonomous, adaptive processes underlying cognitive development.

Acquisition of Reasoning:
A Computational Model of Strategy Development in Transitive Inference

In this paper we want to propose a model of the way basic reasoning processes develop in children. We model the way world knowledge can be used to construct strategies for reasoning, taking account of information processing limitations. We do not assume the prior existence of strategies, but model their development under the dual constraints of conceptual competence and task demands.

The specific focus will be on transitive inference, but the general problem of how reasoning develops will be addressed. Transitivity is being used as a prototypical reasoning task, for several reasons. First, it has the main features that are basic to reasoning. It includes representation, storage, conversion and integration of premises, and entails making an inference based on the integrated representation. Second, there is a very extensive and high quality data base on transitivity, in both the general cognition and cognitive development literatures, which can constrain theory. Furthermore there are a number of good models of the processes entailed in transitive inference (Foos, Smith, Sabol & Mynatt, 1976; Sternberg, 1980a, 1980b), although there is no model of how these reasoning processes develop. There is also less dispute about these processes than occurs in some areas, such as class inclusion or conservation. This provides a solid foundation on which to build.

Acquisition of transitivity has long been regarded as an important developmental landmark. It was regarded as an indicator of concrete operational thought (Piaget, 1950), and has continued to be an important phenomenon for cognitive developmental theories to explain (Halford, 1989). However recognition of the developmental importance of transitivity does not commit us to adoption of Piagetian or any other stage theory. Implications for the stage issue will be considered in the final discussion, but the model does not rest on assumptions of either continuity of discontinuity in cognitive development.

An example of a transitive inference is; if a > b, and b > c, then a > c. Transitivity has a strict mathematical definition1 [1], but the way it is understood psychologically may be different from the mathematical concept of transitivity. The best course therefore is to consider both the mathematical and psychological concepts of transitivity.

To relax the mathematical definition slightly, transitivity means that if R is a transitive relation, and if R exists between a and b, and also between b and c, then the relation R will exist between a and c; i.e. aRb and bRc implies aRc. Examples of transitive relations include those concerned with size, weight, distance, and measurable properties generally. On the other hand "lover of" is nontransitive; if a is the lover of b, and b is the lover of c, it is unlikely that a is the lover of c. [2]

The psychological significance of transitivity rests firstly on the fact that it is part of the definition of the psychologically important concept of serial order. Mathematically, an ordered set is one on which an asymmetric, transitive, binary relation is defined. For example, if we take the ordered set (a, b, c, d), where a is the biggest and d is the smallest, then it will be true that a > b, b > c, c > d, but also a > c, a > d, b > d. The relation "bigger than" is asymmetric (because a > b implies b > a cannot be true), and transitive.

---

[1] A relation R defined on a set S is transitive if aRb and aRc imply aRc for every a,b,c in S.

[2] A relation is nontransitive if aRb and bRc implies aRc for some, but not all, a,b,c. A relation is intransitive if aRb and bRc does not imply aRc for any a,b,c.

The concept of order is important in many situations. Understanding the numbering of houses in a street depends on understanding order, as also does understanding positions in a competition, and a multitude of other everyday concepts. Order is also important to all but the most primitive concepts of quantification. Only the lowest level of scale, the nominal scale, does not depend on order. An ordinal scale clearly entails a concept of order, an interval scale entails the concept of order plus a definition of the distance between the elements and (therefore) an addition operation. A ratio scale entails everything that is entailed in the interval scale, plus a zero point and a multiplication operation. Thus understanding of quantification beyond the nominal scale entails a concept of order, which in turns entails transitivity.

Transitivity tasks are examples of a larger class called N-term series tasks. In general N-term series tasks are presented as a set of premises of the form aRb, bRc, cRd, etc., with a question that asks for the rank order of one or more terms (e.g. which is the largest element, which is larger, b or d?). Our model is designed to apply to N-term series tasks generally.

Models of N-term series reasoning

There are already a number of reviews of work in this field (Breslow, 1981; Halford, 1982, 1989; Maybery, 1987; Thayer & Collyer, 1978), and a number of theoretical models. Two of the best-known models of children's performance on these tasks have been proposed by Sternberg (1980) and Trabasso (1977). Sternberg has offered a mixed model that integrates a number of previous theories, and accounts for a very high proportion of the variance in solution times. The essence of it is that the premises are first processed to obtain the linguistic deep structural base strings that represent their meaning. These strings are then converted to images of ordered pairs. For example, in the problem Tom is happier than Bill, Bill is happier than John, the premises would be coded as the pairs (Tom, Bill), (Bill, Tom). The pivot or common term (Tom) is then located, and the pairs are integrated into an ordered triple, (Tom, Bill, John). Trabasso's (1977) model also proposes that the problem elements are placed in an ordered array, from which the ordinal position of any term can be read.

The model of Foos, Smith, Sabol, and Mynatt (1976) originally applied only to adults, but has recently been extended to children (Maybery, Halford, Bain & Kelly, in preparation). It assumes that the problem is solved by constructing an ordered array in short term memory. The first premise is stored in STM, then an element is sought in the second premise that matches one of the elements in STM, and the new premise is integrated with the one that is stored. Then further premises are processed in the same way, and an ordered string grows in STM. Several operators for performing the integration process have been proposed and empirically confirmed, and these will be considered later.

Sternberg's model applies only to three term series, whereas the Trabasso and Foos et al. models are not restricted in this respect. With the exception of the Maybery et al. (in preparation) formulation, all these models have been applied to problems with premises only between adjacent elements, such as a>b, b>c. Problems with nonadjacent premises (e.g. a>c) have not been considered, although they do have ecological validity, because real-life ordering tasks normally entail nonadjacent relations. For example if we are rank-ordering students according to grade we may know that

Martin is better than Wendy, but it does not follow that Martin and Wendy are adjacent. There may be other students in between. Our model will not be restricted to three-term problems, and will take account of both adjacent and nonadjacent relations.

All these models agree that premise elements are organized into an ordered set, so the raw premise information is not retained. This is also consistent with the fuzzy-set theory of Brainerd and Kingma (1984). Therefore short term memory for the raw premises is not likely to be a major factor in the solutions, but retention of the ordered set should be necessary. The core of the problem is construction of the ordered set. Once this is done, the inference is almost perceptual (Thayer & Collyer, 1978). Therefore the model focuses on construction of the ordered set.

All these models propose a single strategy, which has been treated as the whole explanation for the observed performances. There are two reasons for wanting to go beyond single strategy models. The first is that they beg the question of how strategies originate. Second, it is possible, even likely, that people do not have a ready-made strategy for every conceivable task, or even every N-term series task, but have some general- or domain-knowledge which can be used to construct strategies that meet task demands. Third, the question of how children acquire strategies is vital to understanding cognitive development. Therefore the present paper presents a model of strategy development. In this respect it is allied to models of strategy and skill development in counting, addition, multiplication, and arithmetic word problems, which we will consider next.

Strategy development models

Strategy selection mechanisms can be divided into two kinds, metacognitive and associative. With metacognitive mechanisms the participant makes strategy choices based on an understanding of the problem. With associative selection, the participant has a set of possible strategies, each of which is associated with a particular strength or confidence level that the task can be performed successfully.

The model of Siegler and Shrager (1984) is associative, and has been applied to arithmetic, spelling, word identification in reading, and balance scale tasks. It can be illustrated with addition problems, such as 3 + 3. Strategies are normally tried serially; if the first one fails, the second is tried, then the third, and so on. The first strategy the child tries is to retrieve the answer from memory. There is a distribution of answers that are associated with this problem. The better the answer is known, the more peaked the distribution of associations will be; i.e. the more probable the dominant, correct answer relative to the other answers. When an answer is retrieved, its associative strength is compared with a confidence criterion. If it is above the criterion, the answer is given, otherwise the search of memory continues until the search-length criterion is reached. If no answer exceeds the confidence criterion before the search-length criterion is reached, the child switches to a new strategy. The next strategy tried is likely to be the "elaborated representation". This entails using fingers or images to represent the numbers, thereby adding further associations which might strengthen the association between problem and answer. This strategy is also subject to the confidence and search-length criteria. If it again fails, the child switches to a new strategy, which entails actually counting fingers to determine the answer. The model of Siegler and Shrager (1984), especially in its more developed forms (Siegler, 1987; Siegler & Jenkins, 1989) shows how much can be accomplished through associative strategy mechanisms. However it does not incorporate any mechanisms whereby strategy development can be influenced by understanding of the task. Therefore it seems desirable that it should be combined with some metacognitive selection models that have successfully explained how strategies can be constrained by the performer's concept of the task.

Metacognitive selection mechanisms are based on understanding of the task. The most explicit models of this kind have been by VanLehn and Brown (1980) applied to subtraction, Greeno, Riley, and Gelman (1984), applied to counting, and Greeno and Johnson (1984) applied to arithmetic word problems. In all these models, declarative knowledge is used to construct strategies applicable to particular task contexts. The idea is that people do not have strategies ready-made for every situation, but devise strategies based on their knowledge of the relevant concept and the demands of the task.

The model of VanLehn and Brown (1980) is based on planning nets, which are directed graphs, the nodes of which represent plans for strategies, and the links represent inferences concerning how well each proposed strategy conforms to declarative knowledge. Each proposed strategy is checked against the relevant declarative knowledge, and adjusted if it does not fit. In this way a strategy is devised that is consistent with the participant's knowledge of the task.

The concept of planning net is a sophisticated and powerful one. Van Lehn and Brown (1980) show how a model of the planning process can provide much more fundamental insights into the nature of a task or concept than models based on "surface structure" or the actual procedures used. The reason is that procedures may vary considerably even though the essential concept remains the same. The procedural differences are often trivial or even irrelevant, with the result that models of specific processes may fail to capture the essence of a task. It may be preferable to adopt a model based on the underlying logic of the task. Planning net models enable this to be done.

The model of Greeno, Riley and Gelman (1984) is based on essentially the same philosophy as the planning net models, but differs in the way it is implemented. Its starting point was the observation by Gelman and Gallistel (1978) that young children's efforts at counting, although prone to error, nevertheless reflected an implicit knowledge of the logic of counting. For example, if they counted a set of objects in a straight line beginning with the first one, then were asked to count again making the second object the "one", or even the "two", "three" etc., they could still succeed better than chance. Since these are unconventional counting procedures, it is unlikely that they would have been learned by rote as a skill. The fact that young children were able to perform correctly indicates they understood something about the logic of counting, and could devise a strategy to fit the constraints imposed.

To model this achievement Greeno et al. postulate three kinds of competence, conceptual, procedural, and utilizational. They are defined relative to one another, and the designation of any particular item of knowledge as one type of competence or the other might vary depending how the global task is defined.

Conceptual competence is defined as action schemata that can be used in planning. The action schemata have a degree of general validity; i.e. they are not schemata for specific actions, but really represent understanding of the logic of a class of actions. This understanding can be applied to a variety of different actions, depending on the task demands. An example is action schema 11 (Greeno et al., 1084, p. 113), called KEEP-EQUAL-INCREASE. Its purpose is to ensure that people progress through a set of objects to be counted, and through the list of numbers 1,2,3, . . . n, at the same rate, so that they progress to a new number when and only when they progress to a new object in the set. The prerequisite is that the number of objects already counted must equal the numbers used; e.g. if we have so far counted 3 objects, we must have progressed to number 3. The postrequisite is that this situation must be maintained after the process is completed. The consequence of the schema is that both the set of objects and the set of numbers must be incremented by one in each counting step; e.g., as you progress to the fourth object to be counted, you also progress to the fourth number.

Procedural competence is like the planning heuristics in the model of VanLehn and Brown (1980). It is knowledge about how to assemble action schemata into a strategy. Since every action schema has consequences and prerequisites specified, selection of schemata can be done by means-end analysis. That is, an action schema is chosen whose prerequisites match the current state of the problem, and whose consequences match the current goal. Then another action schema is chosen whose consequences match the prerequisites of the first schema, and so on.

Utilizational competence comprises ability to combine conceptual and procedural competence with information about the task setting. A given set of action schemata could generate many different action sequences, but a sequence must be chosen that is appropriate to the specific demands of the task. Ability to do this is utilizational competence.

Greeno et al. have produced a simulation model that accounts for young children's ability to count, given a variety of situational constraints. It demonstrates a plausible mechanism by which basic knowledge of number can be translated, not just into one counting strategy, but into a variety of strategies that are adapted to the task.

Gelman and Gallistel (1978) postulate that young children understand five principles of counting; cardinality, i.e. the last number reached is the cardinal value of the set; one-to-one correspondence, i.e. every object must be assigned a unique number, and each number used must be assigned to one and only one object; indifference to object order, i.e. it makes no difference in what order objects in a set are counted; stable order of numerals, i.e. the counting numerals must be used in their standard sequence; abstraction, i.e. the objects to be counted need not be of the same kind.

The way conceptual competence is formulated in the Greeno et al. (1984) model implies that these principles are understood in a form that is equivalent to logical rules of general validity. Set theoretic notation is used in the formal specification of the schemata and, while being careful of course not to confuse the terms of the model with the phenomena they are intended to describe, it is still true that understanding of logical, generally valid, principles is attributed to the children.

This model appears capable of accounting for the development of counting knowledge, but if the planning nets approach is to be applied more widely to acquisition of cognitive skills, it seems desirable that a way be found to use it with conceptual representations that are not innate, and are not based on universally valid logical rules. It should be possible for conceptual representations and mental models to be acquired through experience, and the present theory goes beyond existing planning net models in proposing a way for this to be achieved.

It remains to consider the relationship between associative and metacognitive mechanisms of strategy selection. Our proposal is that associative strategy selection is the mechanism of first resort, and it is only when it fails to produce a viable strategy that metacognitive mechanisms are employed. When a task is attempted in a particular context, the strategy which has become dominant through successful use in that or similar contexts, is tried first. If the answers it produces have insufficiently high confidence ratings (or strength, which will be defined later), it is abandoned and the next most dominant strategy tried, and so on. If no adequate strategy is found, then declarative knowledge is examined to see whether a modified strategy can be produced, using planning net-type mechanisms. Thus metacognitive processes are only invoked where usual or habitual techniques fail. In familiar situations we do not think about what to do, we simply perform. The reason is that, although understanding confers immense power for development of new strategies to deal with changed conditions, it is cognitively effortful. Therefore it tends to be used only where habitual strategies are found to be inadequate.

The present model will therefore use associative strategy selection mechanisms as a first resort, and will employ metacognitive mechanisms where associative mechanisms fail. It appears to be unique among contemporary models in blending the two types of mechanism in this way. Our next step is to consider the origin of the conceptual knowledge that forms the basis of metacognitive strategy development.

Origin of conceptual knowledge

Metacognitive strategy development is constrained by the conceptual knowledge of the task. In the planning net model of Greeno et al. (1984) counting strategies are constrained by a concept of number. Strategies for N-term series reasoning should be constrained by a concept of order, because the essence of the task is to assemble the premise elements into an ordered set. We therefore need to consider the nature and origin of the concept of order.

Mathematically an ordered set is defined, as noted earlier, as a set with an asymmetric binary relation. However a more psychologically realistic definition of a concept of order might be expected to have the following features:

Features of a Concept of Order

1. understanding of one or more asymmetric binary relations (e.g. larger than, better than). Note that it is not implied that the child knows the relation to be asymmetric and binary. The child simply knows examples of such relations; i.e. knows that object a can be larger than object b, entailing that b is not larger than a, etc.
2. each element occurs once and only once in an ordered string.
3. end elements have the same relation to all other elements; e.g. a>b, a>c, a>d etc.
4. the position of an internal element is defined by relations to elements on both sides of it; e.g. b<a, b>c.
5. the same relation must exist between all pairs, both adjacent and nonadjacent; e.g. a>b, b>c, . . a>c, etc.

All these features can be instantiated in any ordered set of at least three elements. Therefore a child who has a mental representation of any ordered set of three or more elements has a concrete example of the concept of order. We will show in a later section that a concrete example can be used to constrain performance by using it as an analogy and mapping it into the problem. Therefore a child who has a mental representation of an ordered set of at least three elements has the minimal requirements for a concept of order.

Children have no shortage of experience with ordered sets, because they abound in the environment. Some child-appropriate experiences are order of children in a family, stackable blocks or other toys that vary in size, and positions in a race. We note that the three bears also constitute an ordered set, consisting of Daddy, Mummy and Baby bear. It seems reasonable to assume then that children as young as 3-4 years would have plenty of concrete instances of ordered sets stored in memory. Our model aims to demonstrate that such knowledge can serve as a basis for a concept of order, and can be applied to strategy development through analogical mapping. The model does not assume that the child knows these are ordered sets. All that is necessary is that the child stores a representation of the elements together with the relations between them; e.g. represents at least three objects varying in (a relation such as) size, and knows that object a > object b, object b > object c, and so on.

Origins of manipulative knowledge

The environment also provides abundant experience in manipulating ordered sets. For example it is easy to learn that if three objects (e.g. blocks) are in the order acb, switching the last two results in the order abc. That is, given acb as an initial state, the switching operator applied to b and c produces abc as a final state. We assume that manipulative experience with real objects results in a store of knowledge of the effect of operators on ordered sets in this way. These operators, together with their initial and final states, are the building blocks of strategies.

Application of conceptual knowledge

It is necessary to find a mechanism that would enable experience with ordered sets to be utilized in strategy development. Analogical mapping has the power and flexibility required for this task. According to Gentner (1983) an analogy is a structure-preserving map from a base or source to a target. For example, in the analogy "man is to house as dog is to kennel", "man is to house" is the source and "dog is to kennel" is the target. "Man" is mapped into "dog" and "house" into "kennel". The relation "lives in" between man and house is mapped into the corresponding relation between dog and kennel.

The essence of an analogy is that relations (multi-place predicates) in the source are mapped into the target. Attributes (single-place predicates) are not mapped. For example the attribute of man "wears clothes" is not mapped into dog. Analogies depend on structural similarity between base and target, rather than on element similarity.

An important implication of this for our model is that experiences with quite different kinds of ordered sets can be used as analogies for other ordering tasks. The ordered sets experienced in the past might have quite different elements to those in the task, but this will not necessarily prevent the experience from serving as a source for the task. Element similarity between source and target may facilitate discovery of an analogy, but dissimilarity does not constitute a logical impediment to mapping (Holyoak & Koh, 1987).

Models of analogy

There are a number of process models of human analogical reasoning (Bakker & Halford, 1988; Falkenhainer, Forbus, & Gentner, 1990; Halford, Wilson, Guo, Wiles and Stewart, in press; Holyoak & Thagard, 1989). However only the model of Halford et al is based on parallel distributed processing architectures (PDP); i.e. is connectionist and uses distributed representations. The ACME model (Holyoak & Thagard, 1989) is connectionist, but not a PDP model, because it uses local rather than distributed representations. Connectionist models based on distributed representations have the advantages of graceful degradation and graceful saturation, together with the emergent properties that result from superposition of representations (Rumelhart & McClelland, 1986). These properties make this class of model particularly appropriate to models which want to incorporate realistic processing capacity limitations. The model of Halford et al. has been constrained by evidence of processing limitations, which it makes particularly relevant to our current project.

The STAR analogical reasoning model

Analogies entail mapping base predicates and their arguments into target predicates and their arguments. Therefore the representation of predicate-argument bindings as distributed representations is at the core of a PDP model of analogies. Following Smolensky's (1990) proposal that the variable-binding problem could be handled in terms of tensor products, Halford et al. (in press) showed that predicate-argument bindings could be represented as tensor products of vectors representing predicates and arguments. The resulting model is called the Structured Tensor Analogical Reasoning (STAR) model. The essence of this approach is shown in Figure 1, which also provides a numerical example of a rank-3 tensor product (i.e. with three input vectors).

Consider the simple analogy: woman:baby::mare:foal. This depends on the fact that a similar relation MOTHER-OF exists in the base (with arguments woman and baby) and in the target (with arguments mare and foal). The analogy requires the predicate MOTHER-OF, as well as other predicates which can take woman-baby or mare-foal as arguments, to be represented. The other predicates include PROTECTS, FEEDS, LARGER-THAN etc. Figure 1 shows how this information is represented in the STAR model. There is a predicate vector which represents all the predicates, superimposed. Then there are argument vectors representing the predicates woman and mare (superimposed) and baby and foal (superimposed). The tensor product of these vectors (represented by the activation values within the figure) represents the binding of the predicates to the arguments.

The solution process is shown in Figure 2, using the woman:baby::mare:foal analogy as an illustration. The vectors for woman and baby are used as inputs to the net, and the vector for the predicate MOTHER-OF(-,-) appears as output on the appropriate "side" of the tensor product net. In the next phase of the reasoning process, mare is substituted for woman, and MOTHER-OF(-,-) becomes an input vector rather than an output. The argument 2 slot now produces the output vector, which represents foal. Where ambiguities occur in this process, they can be interpreted by computing the inner product of the output with candidate answers (Halford et al., in press; Humphreys et al., 1989).

Complexity of representations

This approach can be generalized to representations of varying structural complexities. Mathematically, a predicate with two arguments is represented by a tensor product of rank 3: $V_{arg1} \otimes V_{arg2} \otimes V_{pred}$. This approach can be generalized to higher level structures. A tensor product $V_{arg1} \otimes V_{arg2} \otimes V_{arg3} \otimes V_{pred}$ of rank 4 can represent a collection of 3-place predicates. A tensor product $V_{arg1} \otimes V_{arg2} \otimes \cdots \otimes V_{argN} \otimes V_{pred}$ of rank N+1 can represent a collection of N-place predicates. One vector represents the predicate, and the remaining vectors represent the arguments.

A specific example of a rank 4 tensor product would be arithmetic addition. We would have one vector representing the addition operation, and three vectors representing the arguments, which in this case are the addends and the sum. For example the representation of the fact that 3,5 -> 8 under the operation of arithmetic addition would comprise a vector representing addition, and vectors representing 3, 5, 8. The advantage of this representation is that all combinations can be inferred; e.g. given that the operation is addition, and that 3 and an unspecified number x gives 8, the solution to x can be found by the pattern completion process.

The formulation can be linked to a metric for concept complexity, developed by Halford and Wilson (1980) and Halford (1987), based on levels of mathematical structures. The essence of the metric is that the complexity of a concept depends on its *dimensionality,* or the number of independent units of information required to define the concept. The units are of arbitrary size. This is related to the number of arguments in a predicate. A unary relation has only one argument, and therefore has only one dimension of variation. A binary relation has two arguments, and two dimensions of variation, because the values of the arguments can vary independently. Similarly, a ternary relation has three dimensions, and a quaternary relation four dimensions, and so on.

This idea might seem unusual if we are accustomed to thinking of arguments as lists, that can be processed sequentially. However each argument in a relation defines a dimension of variation. The number of dimensions depends on the "-arity" of the relation. A unary relation on a set S is a subset of S. A binary relation on S is a subset S x S of ordered pairs of S. Similarly, a ternary relation on S is a set S x S x S of ordered 3-tuples of S, and so on. Each argument can take a number of different values, so the number of arguments defines the number of dimensions of variation. Therefore the argument sets are not simply lists, but provide a measure of the degree of complexity in the structure.

The number of arguments defines the types of relationships that can exist within a structure. For example, a unary relation, R(x); e.g. BIG(dog) defines an attribute. Because it has a single dimension of variation, it is not possible to specify the way an attribute varies as a function of another attribute. This becomes possible with binary relations, R(x,y); e.g. BIGGER(horse,dog). With ternary relations, R(x,y,z), it is possible to define the way one attribute varies as a function of one other, and how it varies as a function of two others, the latter not being possible with binary relations. As the "-arity" of a relation increases, so do the orders of interaction that are possible. This is a most important feature of predicates for our purposes, because in memory (Humphreys et al., 1989), in cognitive development (Halford, forthcoming), and in the present context of analogical reasoning, we find it necessary to model the orders of interaction among the dimensions of a task.

Functions are a special case of relations; in functions mappings are unique. A zero-variate function, f()=a; e.g. PRIME-MINISTER-IN-1990()=Hawke, has no argument and one dimension in the sense defined above. It is equivalent to a symbolic constant. Univariate functions have two arguments and two dimensions. A univariate function f(a)=b, is a set of ordered pairs (a,b) such that for each a there is precisely one b such that (a,bεf). In a similar way, bivariate functions, f(a,b)=c have three arguments and three dimensions, and so on.

A unary operator has two arguments, and is a special case of a univariate function; e.g. the unary operator CHANGE SIGN comprises the set of ordered pairs (x, -x). Binary operations have three arguments, and have the same dimensionality as ternary relations and bivariate functions. (A binary operation on a set A is a function from the set A x A of ordered pairs of elements of A into A; i.e., A x A -> A. A bivariate function is defined as f:A x B -> C).

The number of arguments in quaternary relations corresponds to the number of arguments in tri-variate functions and in compositions of binary operations. For example, the composition of binary operations of the form a(b+c) = d is defined as the set of ordered 4-tuples {(3,2,4,18), . . . (2,7,3,20). . }.

The number of arguments is related to the dimensionality of a structure because it represents the number of independent terms required to define the structure, and the orders of interaction within the structure. In the context of analogical reasoning, the complexity of a structure mapping task can be related to the dimensionality of the structures being mapped. This will be considered in the next section.

Dimensionality and levels of structure mapping

Four levels of structure mapping were defined by Halford (1987) based on earlier work by Halford and Wilson (1980). These are summarized in Figure 3, together with the tensor product representations they entail. We will consider each level of mapping in turn.

Insert Figure(s) 3 about here

Element mappings are based on 1-dimensional structures, and are defined as:

$$M: R(e_i) \leftrightarrow R(e'_i)$$

It is a mapping between two structures each of which is based on a unary relation. The elements, e., of one structure are mapped into the elements of the other structure, so that the unary relations, R correspond. Because unary relations are predicates with one argument, they correspond to attributes in Gentner's (1983) terms, so mappings based on one-place predicates are validated by attribute similarity. mappings at this level correspond to metaphors.

Relational mappings are based on 2-dimensional structures, and are defined as;

$$M: R(e_i, e_j) \leftrightarrow R(e'_i, e'_j)$$

The elements in structure one are mapped into the elements in structure 2 so that the binary relations in the two structures correspond. At this level there need not be any similarity between elements in the structures, and mappings are validated by similarity of binary relations. Simple proportional analogies of the form A:B::C:D belong to this level. For example, the analogy woman:baby::mare:foal is validated by the similar relation, "mother of" in source and target.

System mappings are based on 3-dimensional structures, and can be defined as:

$$M: R(e_i, e_j, e_k) \leftrightarrow R(e'_i, e'_j, e'_k)$$

Elements are mapped so that ternary relations correspond. At this level there need not be any resemblance even between the binary relations in the two structures. ternary relation, but they may also be expressed as a binary operation, $*(e_i, e_j \rightarrow e_k)$, or as a composition of two or more binary relations, $(e_i \; R \; e_j)$, $(e_j \; R \; e_l)$, $(e_i \; R \; e_l)$. System-mappings permit a high degree of flexibility and abstraction, because they can be used to establish correspondences between structures that have only formal similarities. They also permit analogies to be recognized between superficially dissimilar situations (Halford & Leitch, 1989). However this flexibility and generality is obtained at the cost of higher information processing loads.

Multiple system-mappings are based on 4-dimensional structures, and are defined as;

$$M: R(e_i, e_j, e_k, e_l) \leftrightarrow R(e'_i, e'_j, e'_k, e'_l)$$

Elements are mapped so that quaternary relations correspond. The quaternary relations may be interpreted as compositions of ternary relations, binary operations, or bivariate functions. Like system mappings, multiple system mappings are validated by structural correspondence, and are independent of element or relational similarity. They differ only in the complexity of the structures represented.

In theory mappings based on predicates with more than four arguments are possible, but it is difficult to attach any psychological meaning to such structures, because processing capacity limitations appear to prevent structures of higher dimensionality being mapped.

Capacity limitations

The STAR model is capable of solving analogies that entail three- or four-place predicates. It becomes computationally more costly as we move to higher-place predicates, and consequently to tensor products of higher rank. However this feature provides a natural explanation for increases in processing load that have been observed with predicates of higher rank.

From a review of the literature, Halford et al. (in press) suggest that humans can only process four dimensions in parallel, and they present new empirical evidence to support this hypothesis. They also argue that the number of dimensions corresponds to the number of arguments, because each argument represents an independent source of variation in the structure. It is argued that a tensor product representing a predicate with more than four arguments would be too computationally costly. Tasks which are too complex to be represented in a single tensor product can be handled in one of two ways. These are; ***Conceptual chunking:*** i.e., information can be recoded into a representation with fewer vectors, and ***segmentation***, or dividing the task into into steps that are processed serially. The development of strategies for efficient serial processing is a component of expertise.

An everyday example of a conceptual chunk would be velocity. Velocity is a three-dimensional concept, defined as; velocity = distance/time ($V = s/t$). However it can be recoded as a single dimension, as when (for example) we think of velocity as position of a pointer on a dial. However there is a cost to chunking, because it results in loss of representation of some relationships; e.g. when velocity is chunked as one dimension, the 3-way relation between V, s and/or t cannot be computed without returning to the 3-dimensional representation, which also entails returning to the higher processing load.

Once multiple dimensions are recoded as a single dimension, they occupy only one chunk, and can be combined with up to three other chunks, so velocity can now be combined again with time to give acceleration, ($a=2st^{-2}$), which can then be chunked and combined with mass to produce force ($F = ma$). Thus so we can bootstrap our way up to more and more complex concepts.

This implies that representation of complex concepts depends on availability of efficient codes. Because we can only represent up to about four independent sources of variation in parallel, we require codes that enable us to represent complex concepts without exceeding this limit. One reason why expertise is important is that experts have efficient codes that represent the deep structure of the concept (Chi & Ceci, 1987; Holyoak, 1991). This model emphasizes the importance of efficient coding, and therefore of expertise, in analogical reasoning, because the limitation to 4 dimensions in parallel means that without it complex problems could not be solved.

Representation of transitivity

As noted earlier, a number of contemporary models agree that transitive inferences are made by integrating the premise elements into an ordered string. We suggest that this is a case of analogical reasoning, because it entails using a familiar schema, such as top-down (or left-right) ordering as a basis for organizing the premise elements. The premise elements and relations are mapped into a representation of the top-down schema. This is shown in Figure 4A for the problem; Bill is happier than John, Tom is happier than Bill. The problem elements as assigned to, or "mapped into" the three positions, top, middle, bottom, and there is a structural correspondence between the spatial arrangement and the problem. The top position corresponds to Tom, middle to Bill, and bottom to John.

This is a system mapping, because both structures are based on ternary relations. The mapping is independent of both element and relational similarity, and convention. It is validated solely by structural correspondence.

An example of the type of mapping that children might use is shown in Figure 4C. The base is an ordered set retrieved from memory, consisting of three blocks ordered for size. The hypothetical problem comprises the premises "Tom is happier than John, John is happier than Bill", and the conclusion "Tom is happier than Bill". The analogy is formed by mapping Tom, John, and Bill into the large, medium, and small blocks respectively. The relation "happier than" consistently corresponds to the relation "larger than".

The validity of the mapping does not depend on any resemblance between Tom, John or Bill and the blocks into which they are mapped, or on resemblance between "larger than" and "happier than". The mapping is validated by consistent structural correspondence. That is, each element in the problem is mapped into one and only one element in the base and vice verse (uniqueness), the relations in the source are mapped consistently into relations in the target (larger than is always mapped into happier than), and the arguments of the relations in the source are mapped into the arguments of the relations in the target (in each case, the arguments of larger than are mapped into the arguments of happier than).

The importance of this is that an ordered set stored in long term memory, such as the three blocks in Figure 4C, can be used as a mental model for the ordering required for the problem, purely on the basis of the structural correspondence between the two. In principle any ordered set of three or more elements, stored in long term memory, can be used for this purpose. However there are psychological advantages to sets that do resemble the target, or to sets which are prototypical in the sense that they represent the common properties of many ordered sets, because it is then easier to see the possibility of the mapping being made. However once the mapping has been recognized as possible, similarity does not facilitate the actual mapping process, which can be based purely on structural correspondence, as in Figure 4C (Holyoak & Koh, 1987; Halford, in press).

Once the mapping is made, it is fused into a single representation, the ordered set Tom, John, Bill. The analog provides the structure of the representation, and effectively provides a criterion for the correct way to organize the information in the problem. The fact that Tom, John, Bill is self-evidently the right order to us conceals the fact that at some time children had to come to this understanding. If they do not innately possess logical rules, a likely explanation is that they can use previous experience as an analog.

PDP representation of transitivity

A PDP representation of transitivity, based on the STAR model, is shown in Figure 5A. Because transitivity is a ternary relation, the representation is a rank-4 tensor product. The component relations, aRb, bRc, aRc can be recovered by collapsing over the remaining vector, which is achieved by setting all elements at the value .

The mapping of premises into an ordering schema, as in Figure 4A, is shown for the STAR architecture in Figure 5C. The ordering schema, top-middle-bottom, is shown as the tensor product of four vectors, representing the predicate "Monotonically higher", and the arguments, top, middle, bottom. The problem is represented as the predicate "monotonically happier", with arguments Tom, Bill, John. The two structures are superimposed on the tensor product representation.

Recognition of indeterminacy

Mapping into a mental model consisting of an ordered set of (at least) three objects can also lead to recognition that a problem is indeterminate. Suppose, for example, that the premise "Bill happier than John" were missing from Figure 4A or 4C. It is clear that two different mappings are possible; Bill could be mapped into either middle or bottom, implying the orders Tom, Bill, John or Tom, John, Bill. A mental model in the form of an ordered set can provide a criterion for deciding when another set is ordered. The target set is ordered when it can be mapped in one, and only one, way into the base ordered set.

Relational and system mappings are the levels most relevant to N-term series tasks. The former are involved where premises can be processed one at-a-time, and the latter where premises must be integrated. As we will see, processing premises one at-a-time, without integrating them with earlier premises, produces correct solutions on some but not all problems.

Processing loads of mappings

The four levels of structure mapping increase in abstractness, in that the higher levels are less dependent on element or relational similarity than the lower levels, but this abstractness is obtained at the price of higher processing loads (Halford et al. in press; Halford & Wilson, 1980). The reason is that representations of higher dimensionality entail tensor products of higher rank, which are more computationally costly.

The loads arise from the information that is required to establish that a mapping is valid. With element mappings, elements can be mapped singly, because a mapping is valid if it occurs between similar elements, or between elements linked by a convention. With relational mappings, two elements must be taken into account in each mapping decision, because validity depends on the similarity of the relation between two elements, as illustrated earlier.

With system mappings, validity depends on consistency over two or more relations, which requires that elements be mapped in sets of at least three. This is illustrated by Figure 4B. Notice that, if we consider only two elements in each structure (deleting the element at one end), there is no apparent inconsistency in the mapping. For example "middle above bottom" mapped into "John sadder than Bill", meets the criteria for a valid system mapping. Recognition of the inconsistency requires that all three elements be considered, together with the relations between them.

To see why there is a processing load in such a task, consider what is entailed in the transitive inference "Tom is happier than Bill, Bill is happier than John" who is happiest? We want to assign Tom, Bill and John to the correct ordinal positions. From premise 1 we can only assign Tom to either top or middle position. To decide the correct assignment, we must take account of the other premise also. The same is true for assigning the other premise elements to ordinal positions. Each assignment requires both premises to be considered. That this imposes a high processing load has been empirically confirmed using secondary task indicators by Maybery, Bain & Halford (1986) and Halford, Maybery and Bain (1986). The former study used adult participants, whereas the latter used children. This has the important implication that one source of the difficulty children have with transitive inferences arises from factors that are inherent in the structure of the task, and these factors affect adults also.

Children's ability to perform structure mappings

Research conducted in our laboratory (Halford, 1978; Halford & Wilson, 1980), as well as literature reviewed (Halford, 1982, 1987; in press) indicates that children can perform element mappings by one year of age, relational mappings by two years, system mappings by five years, and multiple system mappings by 11 years (median ages). Furthermore reviews by Brown (1989) and Goswami (in press) support the proposition that preschool children can form analogies if they understand the relations entailed.

The analogies discussed by Brown and Goswami are all relational mappings; i.e. they entail mapping a base into a target in such a way that there is a similar relation in each. Gentner and Toupin (1986) have shown that analogies based on systematicity, which are system mappings, are difficult for children under 8 years. This supports the contention that system mappings occur later than relational mappings. However our success in demonstration system mappings in 5-7 year olds (Halford, 1978; Halford, 1980) suggests that refinement in methodology may show that analogies based on systematicity would be possible earlier than Gentner and Toupin found, but resolution of this question depends on further research.

Children's transitive inference abilities

The question of when children can make transitive inferences has been a major issue in cognitive developmental theory, and there are already several reviews of the literature (Breslow, 1981; Halford, 1982, 1989; Thayer & Collyer, 1978; Trabasso, 1975, 1977). Piaget's (1950) contention that transitive inferences were not understood by young children was challenged by Bryant and Trabasso (1971) who obtained evidence that 3-4 year olds made above-chance transitive inferences. However more reassessments of this research by Halford (1989) and Bryant (1989) have led to the conclusion that there is no valid evidence of transitive inferences before age five. The problem with studies conducted in the Bryant-Trabasso paradigm was that procedures were used which unduly aided children in constructing an ordered set, and that children who were unable to do so were deleted (Halford, 1989). When these factors were eliminated, preschool children failed the task (Halford & Kelly, 1984; Kallio, 1982).

Two main types of reason have been advanced as to why young children find transitive inferences difficult. One is that young children encode premises in absolute terms, whereas older children encode them in relative terms. For example, given the premise "Tom is happier than Bill", young children would encode Tom as happy and Bill as sad. When the premise "Bill is happier than John" is processed, Bill is encoded as happy, contradicting the earlier encoding. This hypothesis has been proposed by Perner and Mansbridge (1983), Siegler (1989), and Sternberg & Rifkin (1979).

The other hypothesis is that preschool children have difficulty integrating premises. For example, Halford (1984) presented ordering tasks that could be performed by considering either one or two premises in a single decision, and found 100 percent success by 3-4 year olds on the former task, but chance performance on the latter. Furthermore Halford et al. (1986) provided evidence that the failure was due to inability of young children to process the loads imposed by premise integration. Halford and Kelly

(1984) required children to learn either overlapping pairs (a>b, b>c, c>d) or nonoverlapping pairs (a>b, c>d, e>f). The former would be easier if integrated, because they would be learned as the string a,b,c,d. If not integrated however the overlapping condition would lead to conflict because of elements b and c being smaller in one premise but larger in the other. Preschool children learned the nonoverlapping pairs but not the overlapping pairs, suggesting that they were unable to integrate the premises.

Recent evidence suggests that four-year olds can integrate premises in certain transitive inference tasks. Pears and Bryant (1990) presented premises in the form of colored blocks placed one above the other; e.g. block A above B, B above C, etc. Childen were required to build a tower with another set of blocks of the same color as the premise blocks, with a top-down order A,B,C,D,E, consistent with the premises. Before building the tower, they were asked inferential questions, such as whether block B would be above or below block D. Four-year olds performed significantly above chance in two experiments, suggesting that they can perform transitive inferences.

Pears and Bryant acknowledge that the task might be performed by manipulating images of the premises. For example, it is possible to imagine pair B-C sitting on top of pair C-D, and it would then be apparent that in a tower B would be above D. This is a legitimate way to make a transitive inference in these circumstances.

The study shows that 4-year old children can integrate premises in certain conditions, but the Pears and Bryant task does not entail mapping from one representation to another. It therefore does not entail anything equivalent to analogical reasoning. In this respect it is instructive to compare Pears and Bryant's procedure with that of Halford (1984). Both used color-coded premises, both obviated the need for retention of premises in memory, and both used tasks that were appropriate for young children. However Halford's task had premises coded in the form of colored pegs in a board. Children were asked to arrange tubes whose colors matched those of the pegs in an order consistent with the pairs of pegs on the board. Pears and Bryant suggest that the difficulties children experienced in Halford's study may have been because it was hard for them to translate spatial position into length.

Actually however this cannot be true in its entirety, because Halford's data showed that even 3-year olds could order the tubes without error, consistent with the pegboard, provided the ordering task was constructed so they could process the premises serially. Therefore they were undoubtedly able to translate the pegboard information into the task of ordering the tubes. What they could not do is make this translation by processing two premises jointly. When they had to order tubes using information from two premises in a single decision, 3-4 year olds failed. Therefore the comparison of Halford's study with that of Pears and Bryant shows that 3-4 year olds had trouble mapping from one representation to another when this mapping depended on two premises, which expressed two relations.

This hypothesis has recently been tested in our laboratory (Andrews & Halford, unpublished). We used the tower of five blocks employed by Pears and Bryant (1990) and an isomorphic sticks task that required children to order sticks from left to right. However to avoid the terms left-right, left was defined as closer to a stuffed toy frog. We also employed two mapping tasks, in which children had to use pairs of blocks to determine the order of sticks, or vice verse. The inference task based on mapping requires children to map two premises jointly, whereas the corresponding construction task can be performed by processing the premises one at a time. If Pears and Bryant's demonstration of transitive inference is valid, then by Postulates 3.0 and 5.1, and from the argument above, it would be predicted that children under five years would succeed on all tasks except the inference question based on mapping, because this is a system mapping, and entails mapping two relations jointly.

Pears and Bryant's finding of above-chance transitive inference in 3-4 year olds was replicated for the blocks task, but not for the sticks task, which suggests it is not very robust. The prediction that 3-4 year olds would fail on the inference task based on mapping was confirmed, even though they succeeded on all construction tasks, showing that they could map one relation from blocks to sticks or vice verse. Despite the great ingenuity shown by Pears and Bryant in devising a transitive inference task that is eminently suitable for young children, we still find severe limits to their ability to process two relations.

There seems to be good evidence that both encoding and integrating are processes that improve with age, but it seems unlikely that young children are completely unable to encode relations. Both Bullock and Gelman (1977) and DeLoache (1989) have provided evidence that 2-3 year old children encode relations between objects. For example, DeLoache showed young children could locate a hidden toy in a room after seeing a model toy hidden in a model of the room. Presumably this task entails encoding the relation between the model toy and a model piece of furniture (e.g. behind the lounge), then map this relation into the real room. It is a good example of a relational mapping task (Halford, in press). There does seem to be good evidence however that children under age five years have difficulty integrating relations. Progress in transitive inference and ordering between approximately two- and seven years depends to a considerable extent on switching from processing relations independently, to processing them in an integrated way. This will therefore be one of the major developments that the model will attempt to simulate. As we will see, integrated representation of the premises can be expected to result in a shift towards relational encoding. Therefore the question of whether development of transitive inference depends on relational encoding or on premise integration may turn out to be a non-issue.

Working memory architecture

Baddeley (1986, 1990) and Schneider and Detweiler (1987) have provided sophisticated reviews of the working memory literature. It is clear that working memory consists of more than one system, and that the aggregate capacity is much greater than the small number of items in short term memory span. Our model does not make strong assumptions about working memory limitations, but it provides for this factor to be manipulated.

Outline of the model

The model has the following major objectives:

1. To simulate development of N-term series strategies, under the constraint of a concept of order that could be acquired through experience.

2. To integrate associative and metacognitive strategy selection mechanisms.

3. To simulate a major acquisition in middle childhood, the ability to integrate premises.

Main features of the model

1. Transitive inferences are made by constructing an ordered set of premise elements, as in models of Sternberg (1980) and Foos, Smith, Sabol & Mynatt, (1976). The ordered set, called a "working memory resultset" is stored in short term memory.

2. There are no strategies as such, the role of strategies being filled by productions.

3. If productions exist that match the current state, they are invoked without metacognitive processing. If no productions match the current state of the problem, new productions are created by metacognitive processes, constrained by the concept of order.

4. It is assumed that operators are learned by manipulative experience, and are stored in long term memory, together with initial and final states, so they can be used in means-end-analysis. An example would be the "switch" operator (Table 1), which reverses the order of two elements. It is assumed that children have learned during manipulative play that if two objects are in the order AB, and they are switched around, that the order will be BA. Thus the initial state is AB and the final state is BA.

5. It is assumed that a concept of order is acquired through experience (such as play with blocks that form series), and can be instantiated as an ordered set of at least 3 elements, stored in long term memory. The essential features of the concept of order were outlined above.

6. Analogical mapping of the ordered set in long term memory into the strategy output is used to assess validity of strategies. This constrains the strategies to produce an output consistent with the concept of order. This mapping produces a high processing load, which causes difficulty for children and adults, but the magnitude of the difficulty is greater at younger ages.

7. Strength of productions is increased if they produce correct orders, decreased if they produce incorrect orders (associative strategy development).

8. Negative feedback leads to increased effort, which leads to higher level structure mapping processes being invoked.

9. If relational level structure mapping is invoked, strategies are produced that do not integrate premises. If system level structure mapping is invoked, strategies are produced that do integrate premises.

10. Strategies that do not integrate premises produce correct solutions on some problems, but produce errors on other problems. This leads to negative feedback, which increases effort, which leads to system mapping being invoked, and strategies are then developed that do integrate premises.

Architecture

The model was programmed in the production system language PRISM II (Ohlsson & Langley, 1986). The whole model described in this chapter is based on productions. The program does not have access to the PDP based STAR model of analogical reasoning described earlier. Instead, structure mapping is carried out by productions which perform the same functions as relational- and system mappings specified for the STAR model.

A production system is composed of a set of condition-action pairs each of which specifies that if a certain state occurs in working memory, then a mental or physical action should be performed. For example, the traffic rule "stop on a red light" could be coded as a production of the form;

if a traffic light is red --> stop

The left-hand-side of the production refers to a condition; in this case, a red traffic light. The right-hand-side refers to an action; in this case stopping. When productions systems are used to model problem solving, the actions are usually mental; e.g. entering information in, or retrieving it from, working memory, manipulating information in working memory, setting goals, etc.

PRISM II is based on the ACT* model of cognition (Anderson, 1983), and the model we present has a number of features in common with Anderson's approach. Some of these are structural, such as the use of productions and hierarchical goal structures (Figure 6). However more significant similarities occur in the way strategies or cognitive skills are acquired. The current model, like that of Anderson (1987) proposes that strategies are first acquired through domain-independent, general processes such as analogical reasoning and means-end analysis (the so-called "weak" methods). Once acquired, strategies are strengthened or weakened according to their success in solving problems. However PRISM II has many parameters and structures that can be altered by the programmer, so it does not constrain the model builder to adopt the assumptions of the ACT* theory. Furthermore no attempt has been made to either conform to or depart from Anderson's theory, and the model has been constrained solely by data on N-term series problems and other relevant cognitive phenomena.

The productions that have been programmed into the model are basically domain-general, in the sense that they are not specifically related to N-term series problems. One set performs such functions as setting and removing goals, requesting feedback, starting new problems, and building new productions. Another set searches for elements, compares elements, and codes elements into features. Domain-specific productions that are specialised for performing N-term series problems are built by the model in the course of solving problems.

There is a working memory which stores the current state of the problem, including goals. This information is subject to decay over time. When items are added to working memory they are given an initial activation value of 1.0. Activation decays on subsequent cycles, delta x = -0.1x. However goals are not subject to decay, because it is assumed that goals have highest priority, and will always be maintained.

Domain-general productions are stored in a procedural memory, and their strength does not vary. It is assumed that their use has been so general that their strength will affected negligibly by processing of these problems. Domain-specific productions that are built in the course of problem-solving are stored in new-procedural memory, and have a strength which varies in two ways. First, strength is increased following successful performance of a problem according to the formula;

$$\text{delta } S+ = .1(1-S),$$

and decreased following unsuccessful performance, according to the formula;

$$\text{delta } S- = .25S.$$

Second, there are small random fluctuations according to the formulas;

$$\text{delta } S+ = v(1-S)$$

$$\text{delta } S- = vS$$

where v fluctuates randomly between 0 and .2.

This causes strength values to asymptote at either 0 or 1. The random fluctuations cause strength values to tend towards a common value of .5.

There is also a declarative memory which stores the goal hierarchy, and the information about conditions and consequences of operators. The operators are listed in Table 1. As noted earlier, it is assumed that these operators have been learned through manipulative experience, such as play. For example, the append operator has preconditions that object 2 is not in the list, and object 1 is in the list. The operator has the effect that object 2 is appended to object 1 in the list. This is assumed to be a primitive function that would have been learned manipulatively.

Insert Table(s) 1 about here

The effect of different memories is to categorize contents so that, for example, declarative memories contain different kinds of information from procedural memories. This is a matter of programming efficiency, and does not imply that these types of information occupy physically different memories in a human being. The categorization function in human memory is more likely to be achieved by linking stored information to context (Humphreys, Bain & Pike, 1989; Schneider & Detweiler, 1987), but computer-simulation of this process requires a neural net architecture. Integration of neural net and production system architecture is not yet practicable, at least on the scale of this model, although progress is being made in this direction (Touretzky & Hinton, 1988).

Operation of the model

Overview We will illustrate the model using a sample problem as shown in Figure 7.

Insert Figure 7 here

The first premise is a>b. The model checks whether there is an ordered string already in working memory, finds there is not, and adds the string ab to working memory. This can be done because the model is set up with domain-general productions for storing information in working memory. When the premise b>c is presented there is no production that has a string in working memory plus a premise, in its condition side, because the model is not equipped initially with productions for dealing with transitive inference.

This puts the model into a modification phase. First it searches for a concept appropriate to its present goal, which is ORDER OBJECTS. This causes the concept of order to be retrieved from memory. Analogical reasoning is used to determine the correct order, by mapping the premises a>b and b>c into an ordered set retrieved from memory, as explained previously. It is recognized that the ordering abc would be consistent with the concept of order. Means-end analysis is then used to select an operator which will produce the order abc. Then a new production is built which includes the APPEND operator, as shown in Figure 8.

Insert Figure 8 here

This production will match any state in which working memory contains one or more objects in a string, and the first object in the premise matches the last object in the string. On the action side, the APPEND operator will place the second object in the premise on the end of the string.

When a subsequent problem occurs which requires this production it will fire, without the effortful process of having to build it. Thus strategies, which in this model consist of productions, can operate in relatively automatic fashion once they are acquired. Understanding is not required for problems of a type which the model has already learned to solve.

When a correct solution is obtained, positive feedback increases the strength of the productions involved in the solution. Negative feedback results in weakened productions. Productions that lead to errors are gradually weakened below threshold, and cease to fire. This clears the way for construction of new productions that avoid the errors.

When the model first builds productions, some of them do lead to errors. Some of these errors occur because of the processing load imposed by system mappings; i.e. mapping both premises into an analog, as in Figure 4. This can result in a tendency to focus on the most recent premise, ignoring the implications of earlier premises. We call this the relational strategy, because it uses only relational mappings, as defined earlier, and processes only one relation at a time. The operation of relational and system strategies are illustration in Figure 9.

Insert Figure(s) 8 about here

The first premise, c>d results in the string cd being stored in working memory. The next premise, a>b contains no element which matches any element in cd. If the model is operating at a low level of cognitive effort, either because of low arousal, competition for resources, or inadequate processing capacity, then it pays attention only to the most recent premise, and performs a relational mapping. That is, it maps a>b into the analog of an ordered set, ignoring c>d. Therefore the production that is built simply inserts ab after cd, yielding the order cdab, without recognizing that the order of the pairs is indeterminate. When b>c is presented, attention is again paid only to the most recent premise, resulting in a production which simply switches b to the front of the list, yielding the order bcda. Data from Foos et al.'s (1976) study, and from our own laboratory, show this is a common error for children, and for adults under high processing load.

This error leads to negative feedback, which weakens the productions involved in the incorrect solution, clearing the way for new productions to be built. The negative feedback also increases effort, which causes the most recent premise to be integrated with previous premises when a new production is being built. This leads to a production based on system mapping as defined earlier.

Now when a>b is appended to the string, a marker is inserted to indicate that the order of the pairs is indeterminate. Then when b>c is presented, a production is built that switches ab to the front of the string, yielding abcd. Correct feedback strengthens this production.

The model makes these errors when it first builds strategies, but then builds new productions which provide correct solutions to problems with all permutations of premises. It also recognizes when a problem is unsolvable, because it codes indeterminacies in the ordered set in working memory.

Productions

The productions that are built into the system before it begins operation are listed in Appendix A. The productions that are built in the course of the run to be described in this section are listed in Appendix B.

Two parameters are defined, effort level and strength of productions. Effort level affects the type of structure mapping that is performed. Because structure mapping based on two relations imposes a high processing load and is therefore cognitively effortful (Kahneman, 1973), participants tend to avoid using it. Therefore people tend to map using only one relation, usually the current premise. This produces correct performance some of the time, but does not yield valid and reliable premise integration. In our model, effort is set initially at a level that leads to mapping using only one relation. This leads to errors, and the resulting negative feedback causes effort to be raised to a level that produces mapping based on two relations.

When effort is below threshold, only relational mappings are performed, when it is above threshold, system mappings are performed. On this run the threshold level was set at 1.30. Strength affects the likelihood of a production firing. As is standard in production systems, a precondition for a production to fire is that its conditions match elements in working memory. If more than one production matches, the one with the greatest strength fires.

The strength of productions involved in producing a solution is increased when it is correct, and decreased when it is incorrect. The names of productions that have contributed to a solution by altering the working memory resultset are placed in a list. Following feedback, the strengths of all productions in the list are adjusted according to the formula given in the previous section. Productions that are below threshold (initially .50 on this run) will not fire. There is also a random component in the strength parameter, which varies according to the formula given in the previous section. This has the effect of causing differences in strength to be gradually reduced, thereby permitting productions that have recently been weakened to eventually reappear.

Run of the model

The model was run on the set of problems shown in Table 2. The first 6 problems entail adjacent and nonadjacent relations, and consist of all permutations of the premises a>b, b>c, a>c. Problems 7-12 do not entail nonadjacent premises, but contain a fourth term, and comprise all permutations of the premises a>b, b>c, c>d. The value of the fourth term is that it requires the system to iterate the ordering processes. For example, having integrated b>c with a>b, producing abc, it then must integrate c>d with abc to produce abcd. A model that can do this can produce series of any length, provided they are based on adjacent relations only, and subject to a memory capacity that is adequate for storing the resulting string. The problem is different however with nonadjacent relations, because series of more than 3 terms cannot be constructed simply by iterating processes used in 3-term problems.

Insert Table(s) 2 about here

Pilot work showed that series of more than 3 terms with non-adjacent relations cannot normally be handled by human participants without external aids. The fact that the model can handle series of any length with adjacent relations, and can process adjacent and nonadjacent relations, gives it a high degree of generality. The generality is increased by the fact that elements and relations in production rules are defined as variables. This means that in principle any type of element or relation can be handled. However specific instantiations will be discussed in the run to be described, for the sake of clarity.

When the operation of the model begins, the goal "order objects" is set. This is equivalent to participants attempting to perform a N-term series task. The next production requests instructions, which can be one of "stop", "restart new problem" or "add a premise". The instruction can be supplied interactively by the user, or read in from a file. If the instruction is "stop", the FLAG "halt" is set, so no new productions will be fired, and the goal "order-objects" is deleted. If the instruction is "restart new problem" the initial goal is reset, and further instructions are requested. If the instruction is "add a premise", then a goal is set to integrate that premise into the working memory resultset.

If the working memory resultset is empty, then the elements of the new premise are simply placed in working memory in the correct order. For example, if the first premise were "a>b" then "ab" would be placed in working memory. Then the next premise is requested.

If there is already a premise in working memory, then one or more "match" productions are fired. These productions seek elements that are common to the working memory resultset and the premise. For example, if "ab" were in the working memory resultset, and the premise were "b>c", then the match element would be "b". The "matchlist" is then placed in working memory.

This is as much as can be done using productions that are "prewired" into the system. That is the system has productions for obtaining and comprehending premises, storing and retrieving information in working memory, and comparing new premises with the working memory resultset. It can identify elements common to the new premise and the working memory resultset, and mark the position of the common elements. For example, with "ab" in the working memory resultset and "b>c" as the new premise, the common element is "b" and it is at the "back" of the string in the working memory resultset. Strategies for proceeding beyond this point are constructed by the system. Once constructed they are stored and can be used on future problems, without having to be constructed again.

The comprehension of premises is not modelled in detail, it being assumed that the system is capable of extracting the semantic information in the premises and storing it in STM, or using it in construction of ordered sets. The reason is that this aspect of the task has already been well modelled in Sternberg's (1980a, 1980b) formulation, and in this project we wanted to concentrate on the development of strategies for premise integration. This formulation is quite compatible with Sternberg's account of premise comprehension, and if the premise comprehension process could be simulated, in principle it could be integrated with our model.

A run of the model through the 12 problem forms over four cycles is shown in Table 3. An overview of the run will be given before considering the details. Additional runs have been made with problems in random order and the outputs have been essentially the same. The main difference was that in some cases strategy shifts occurred more readily. This occurs when a number of problems happen to occur close together that cause incorrect productions to be weakened. However for convenience we will examine two runs of the problems in the order shown in Table 3. The effort level is initially set low (.6), so relational structure mapping will be used.

Insert Table(s) 3 about here

On the first cycle the model is building productions to deal with N-term series problems. Effort level for structure mapping is initially low because of the high demands of other processes, and because the need for system mappings is not recognized. The first set of productions built produce correct responses to most problem forms. However, because of the low effort level, incorrect orders are produced initially on those problem forms which require premise integration. Specifically, on problem forms 1 and 2, the incorrect order acb is produced. These are errors that were commonly found in previous research (Maybery, Halford, Bain, & Kelly, in preparation). The former was explained in an earlier section. The latter results from first combining b>c and a>b to produce abc, then when a>c is presented, c is placed after a, yielding acb. As with the first problem form, this error results from processing the final premise by itself, without integrating it with earlier premises. The order produced is incompatible with the second premise, a>b, but this is not recognized by the model at this point in its development, because it only maps premises into the concept of order one at-a-time.

The error on problem 1 results in negative feedback, which reduces the strength of the productions which fired for that problem, and increases effort level to so the model will continue to operate at the relational level, resulting in further errors on some problem forms.

On problem 2, a new production is built to process the second premise, because (a>b) has not previously been processed following (b>c). The resultset after the second premise will be a,b,c, as with problem 1. When (a>c) is presented, the production which processed it with the same conditions in problem will have been weakened, below threshold. However the model is not yet ready to build a system level production. To avoid rebuilding failed productions, or abandoning problems because no adequate production can be built, the model now checks whether a production has previously been built with the same conditions and actions. Such a production exists; token-2. The threshold is temporarily lowered, to permit the production to fire again, resulting in the same error as before. The effort level has also been increased in the search for a production. Thus there is competition between processes which lead to new productions and processes which reinstate old productions.

The error on problem 2 raises the effort level to 1.409, which allows system level productions to be built. On problem 3, premise (a>b) is processed as before. In an attempt to build a production to process premise (a>c) system mapping is employed, and a>b,a>c are mapped into the concept of order. It is then recognized that the order is indeterminate, so a production is built which inserts an indeterminacy marker, <<c ^ b>>, showing that the order of c and b is indeterminate. This is resolved by premise (b>c), yielding the resultset, abc.

The remaining problems are processed using system level mapping, resulting in productions which take account of the current premise and the most relevant, previous premise. This means that indeterminacies are noted, where the occur after the second premise, and the remaining problem forms are processed without error. On cycle 2, all problem forms are processed without error. The sequence of productions which fire on problem forms 1-3 in Table 2 is given in Appendix C.

To summarise the run of the model, it began with domain-general productions, then built domain-specific productions appropriate for the N-term series problems presented. Metacognitive processes, which entailed mapping premise information into a concept of order, instantiated as an ordered set of three elements, was used to guide the creation of new productions. Once created, productions were subject to strengthening or weakening by associative learning mechanisms, depending on their success in producing correct problem solutions.

Experiments

We will now consider 5 experiments designed to test children's ability to perform a number of functions predicted by the model. Experiment 1 tests the prediction that children should be able to map one ordered set into another. Experiments 2 and 3 assess children's ability to recognize when the order of a set of elements is determinate. Experiment 3 also examines the orders which children produce as premises are presented successively, so these can be compared with the orders predicted by the model. Experiments 4 and 5 assess the processes children use to solve N-term series problems, so these can be compared with the processes incorporated into the model.

Experiment 1: Mapping ordered sets

In this experiment children were shown an ordered set as standard, and required to say which of two comparison sets, one ordered and one not, was like the standard. This required children to recognize the correspondence between two ordered sets, which is equivalent to mapping one ordered set into the other. *Task overview*

Each trial consisted of three cards, as shown in Figure 10A or 10B. The bottom card was the standard, and the two cards above it were the comparisons. The participant's task was to say which comparison card was similar to the standard in a specified way. In each trial the standard varied in one dimension, which we call the primary dimension. The other dimension which varied in some trials is the secondary dimension. For example in Figure 10A, the standard is ordered with respect to height, and the correct comparison (left) is ordered with respect to both height (primary) and width (secondary). The incorrect comparison is ordered with respect to neither dimension. Figure 10B is a transfer task, and the correct comparison (right) is ordered with respect to width but not height.

To ensure the selection could not be made by choosing the card with elements similar to the standard, on two thirds of test problems the elements also varied on a secondary dimension (as illustrated in Figure 10A). When this was done, the correct card was ordered with respect to primary and secondary dimensions, so as to avoid a conflict of two orders. An additional purpose of variations in the secondary dimension was to show children that elements could be ordered with respect to other dimensions, thereby easing the transition to the transfer items.

In the transfer problems, the correct comparison item is ordered with respect to the secondary but not the primary dimension. In the transfer problem in Figure 10B, the standard is ordered according to height, and the correct comparison is ordered according to saturation. None of the comparisons were ordered with respect to the primary dimension in transfer problems.

Introductory problems were used to familiarise participants with the idea of ordering from left to right. They comprised three cards, each with two elements; e.g. a taller object on the left and a shorter object on the right. Children's attention was drawn to the fact that in the standard the taller object is on "this side" (indicating the left side of the standard card). They would then be asked to say which of the comparison cards had the taller bottle on that side. The purpose was to teach the idea of ordering elements from left-to-right, and to introduce the dimensions of height, width and saturation used in the test problems.

Method

Participants

There were 24 children, 9 males and 15 females, aged 5-1 to 5-10 (mean 5-6) from three middle-class preschools, and 24 children, 10 males and 14 females, aged 6-9 to 7-11 (mean 7-5) from a middle-class school.

Materials

The three dimensions used were height (h), width (w), and saturation (s). The six combinations of primary (given first) and secondary dimensions were h x s, h x w, w x s, w x h, s x h, s x w. Each combination was used to form one introductory, one test and one transfer problem.

Height of figures varied from three to six cm., width also varied from three to six cm., and saturation varied from a bright, intense, highly saturated color to that same color fading towards white. The colors were cut from Pantone color paper/Graduated/Uncoated.

The cards for the introductory problems averaged 41 square cm, and had grey backgrounds. The two comparison cards were placed approximately 5 cm apart, with the standard card approximately 1 cm below their lower edges. The figures on the cards were bottles, drums, or rockets, colored red, blue, or green. The same object and same color were used throughout any one display.

The cards for the test and transfer problems averaged 105 square cm, and were similar in background and positioning to the introductory cards. The figures were mushrooms, houses, and toffees.

Procedure

Each participant received the following sequence:
1. three, two-trial introductory problems
2. one three-trial practice problem, using the test problem format
3. three, three-trial test problems
4. one three-trial practice problem in the transfer problem format
5. and three, three-trial transfer problems.

The same standard was used for all trials of a problem, but the comparisons varied. Each set of three problems was based on one of two combinations of dimensions: w x s, h x w, s x h, or h x s, s x w, w x h. Half the participants received one combination, half the other, at random. This ensured that each dimension was used once as primary and once as secondary for each problem set for each participant. Order of problems was randomized.

Referring to the example introductory problem mentioned earlier, children were asked to point to the taller bottle and to the shorter bottle in each card. Synonyms (e.g higher, lower) were permitted where there was no possibility of confusing dimensions. Then their attention was drawn to the position of the taller bottle on the standard card, and they were asked to select the comparison card that had the taller bottle in the same position. Feedback was given.

In test problems the child's attention was drawn to the fact that the elements were ordered in the standard, and (referring to the example in Figure 10A) was asked to point to the tallest, the next tallest, and the shortest. The child was then asked to select the comparison card that was also in order. For the practice problem immediate feedback was given. On test problems, children were asked to select the comparison card that was like (ordered the same way as) the standard. Feedback was given after the three trials were complete. This would help the child to acquire the concept of order and facilitate later problems, without interfering with responses within the problem. Participants were required to answer five of the last six trials correctly in order to proceed to the transfer problems.

A transfer problem was preceded by three trials of a test problem using the same primary and secondary dimensions. Otherwise the procedure for the transfer problems was the same as for the test problems.

Results

The mean number of correct test-problem responses (out of 9) was 7.25 (S.D = 1.92) for the five-year olds, and 8.63 (S.D. = .70) for the seven-year olds, $t(46) = 3.23$, $p < .01$. Both means are significantly higher than the chance value of 4.5, $t(23) = 7.00$ for the five-year olds, and 28.43 for the seven-year olds. Although there is an age-effect on overall accuracy, both groups can discriminate ordered from unordered sets at better than chance level.

All the seven-year olds but only 18 of the 24 five-year-olds met the passing criterion of five correct responses out of the last six. Those five-year olds who failed did not perform better than chance (mean correct = 4.50, S.D. = 1.12).

On transfer problems, the mean correct (out of 9) for the five-year olds was 7.56 (S.D. = 1.80) and for the seven-year olds was 8.08 (S.D. = 1.41). The age effect is not significant, but of course only the successful five-year-olds progressed to the transfer task. Both groups performed significantly better than chance, $t(17) = 7.19$ for the five year olds and $t(23) = 12.40$ for the seven-year-olds, $p < .001$ in both cases. Therefore all participants who passed the test items transferred successfully.

The appropriate conclusion is that all the seven-year old sample, and 75 percent of the preschool five-year old sample can discriminate between ordered and unordered sets. Both 7-year olds and 5-year olds who can pass the test problems can transfer the discrimination from one dimension to another, which shows that the discrimination is not based on specific features of the stimuli.

Discussion

The transfer task requires children to recognize the correspondence between one ordered set and another, even though there may be little similarity between the elements and relations of the two sets. This can be treated as a structure mapping, as shown in Figure 11. The standard set in Figure 10B corresponds to the correct comparison set in that figure because the tall figures maps into the narrow figure, the medium height figure into the medium width figure, and the short figure into the wide figure. The relation "higher than" consistently corresponds to "narrower than". There is a structural correspondence between the standard and the correct comparison. This does not exist however between the standard and the incorrect comparison, as shown in Figure 11B. Here the mapping is not consistent, because "higher than" in the standard is mapped into "higher than" in the comparison on two occasions, but into "shorter than" on the third occasion. The inconsistent mapping reflects the lack of consistent correspondence between the standard and the incorrect comparison set. The mapping is a system mapping because it depends on consistency and uniqueness, rather than on element or relational similarity. As explained earlier, construction of an ordered set depends on system mappings.

The finding that 75 percent of five-year olds succeeded in the transfer task supports a central contention of the model, that five year old children, who can perform transitive inferences, are capable of mapping one ordered set into another. Analogical mapping of ordered sets is a core process in the model, and it constrains strategy development. Experiment 1 indicates that the model is realistic in postulating that children can use analogical mapping of one ordered set into another.

Recognition of indeterminacy

Another prediction from the model is that children should be able to recognize when the order of premise elements is indeterminate. As explained earlier, they can map the premises into an ordered set representing their concept of order. Indeterminacies can be recognized if there is more than one way the mapping can be made. Furthermore the missing premises can be recognized because there will be relations in the concept of order that do not have corresponding relations in the premises.

Consequently Experiment 2 will assess children's ability to recognize when the premises are sufficient to determine the order of the elements. Two premises will be presented for each problem which will be either determinate or indeterminate. Participants will be asked to judge determinacy after both premises have been presented, and will be asked to construct orders only for problems that they judge to be determinate.

Experiment 2

Method

The participants were from a primary-school in a lower-middle class suburb of Brisbane, Australia. The 10 younger participants had a mean age of 7 years 11 months (range 7-5 to 8-5). The 10 older participants had a mean age of 10 years 2 months (range 9-7 to 10-7).

Apparatus consisted of a BBC model B microcomputer, connected to a 23 cm Amust color monitor.

Problems. Two determinate and two indeterminate problem forms, each with two premises, were used. The determinate problems were A>B,B>C and B>C, A>B. The indeterminate forms were A>B,A>C and B>C,A>C.

Thirty-two problems were generated for each participant. The eight practice problems used each of the 4 problem types twice, and the 24 test problems used each problem type six times. The test problems were divided into two equivalent blocks, each of 12 problems.

Three of the digits 1-9 were randomly selected as the A, B, and C terms on each problem. One restriction on the selection was that numerically-adjacent digits could not take neighbouring positions in the ordering. The digits were printed on animated runners that "ran on the spot". This permitted a concrete description of the task, for example, "runner 4 is ahead of runner 6, and runner 6 is ahead of runner 2". The runners were 3 cm high, and colored red and yellow with white 1 cm numerals.

Procedure.

Each problem was initiated by the participant pressing the space bar, which was labelled NEXT. This was followed by 3s of blank screen, after which the first pair of runners was printed centrally on the monitor. The participant proceeded to the second pair by pressing the space bar. The previous pair was obliterated when the new pair appeared. Perusal time for each pair was recorded to centisecond accuracy.

On practice problems, feedback consisted of a simultaneous display of (1) all of the pairs, (2) a row of asterisks marking where the sure button should have been pressed, (3) the participant's ordering, (4) the correct ordering, and (5) a typed message indicating which components of the task (pressing the sure button and providing the ordering) were correct. This feedback continued on the test problems, except on trials where both components of the task were completed successfully, in which case a message to that effect was printed.

Initial instructions familiarized the participant with the relevant keys on the keyboard, including the DELETE key, used for correcting answers. Then the Experimenter worked through a simple problem on paper, and the first practice problem was solved with the Experimenter recording the successive pairs on paper. The participant was permitted to record the pairs on paper on the second problem, but the pencil and paper were removed for subsequent trials. The Experimenter provided detailed feedback on the practice problems.

After the second premise had been understood (indicated by pressing the space bar), the prompt "your answer" appeared on the screen. Participants were instructed to press the return key, marked with an asterisk, and referred to as the "can't tell" button, if they thought there was more than one correct order. Otherwise they were asked to type in the correct order. As each number was typed, its corresponding runner was printed on the screen. Participants were tested individually in a single 20-30 minute session. Errors were corrected with the DELETE key.

Results

Three variables were recorded:

(1) Determinacy judgment. Judgments were scored correct on indeterminate problems if participants indicated they could not order the elements, and on determinate problems if they attempted to specify the order, irrespective of whether the order they provided was correct.

(2) Problems correct. Scoring for indeterminate problems was as for (1), but on determinate problems the order provided by participants had to be correct for the response to be scored correct.

(3) Second premise latency, i.e. time to inspect the second premise. Second premise latencies reflect variations in processing between different problem forms of the problem, whereas first premises do not discriminate between problem forms. Maybery et al. (in preparation) showed that second premise latencies were most revealing of processing differences.

Each ANOVA comprised the factors age and problem type. The latter factor was partitioned by planned comparisons into determinate versus indeterminate. Determinate problems were partitioned into a>b,b>c versus b>c,a>b, and indeterminate problems were partitioned into a>b,a>c versus b>c,a>c. Block was included in preliminary analyses and only one significant effect was found, an interaction of problem form by block on judgments of determinacy, $F(3,54) = 3.63$, $p < .05$. This was due to a small reduction in performance on a>b,b>c problems in the second block. Because it does not modify the interpretation of any other effects of interest, it will not be discussed further.

Determinacy judgments yielded an effect of determinate versus indeterminate, $F(1, 54) = 6.40$, $p < .05$. The proportion correct was .87 for determinate problems, and .75 for indeterminate problems. A second effect was a>b,a>c (.83 correct) was easier than b>c,a>c (.67 correct), $F(1,54) = 6.61$, $p < .05$. There was no age effect.

In order to test whether children were significantly discriminating between determinate and indeterminate problems, a planned comparison was made of determinate versus indeterminate problems on the number of "can't tell" responses. This yielded $F(1, 54) = 132.49$, $p < .00$. The proportion of can't tell responses was .75 for indeterminate problems and .13 for determinate problems. Therefore the children were discriminating between determinate and indeterminate problems very reliably.

On problems correct determinates (.51 correct) were harder than indeterminates (.75 correct), $F(1,54) = 16.27$, $p < .001$, but this only reflects the more stringent criterion for determinates, where the correct order was required as well as a correct determinacy judgment. Two marginal effects were that a>b,a>c problems (.83 correct) were easier than b>c,a>c problems (.67 correct), $F(,54) = 4.01$, $p = .05$, and a>b,b>c problems (.59 correct) were easier than b>c,a>b problems (.43 correct), $F(1,54) = 3.62$, $.05 < p < .10$. There was an effect of age, $F(1,18) = 8.15$, $p < .05$. For the seven-year olds the proportion correct was .56 and for the 10-year olds it was .70.

Second premise latency was shorter for indeterminate problems (5.60 seconds) than for determinate problems (9.50 seconds), $F(1, 54) = 38.55$, $p < .001$. This is consistent with Sternberg's (1981) model. There was a marginal age effect, $F(1,18) = 3.41$, $.05 < p < .10$. The seven-year old mean was 8.73 seconds, and for 10-year olds it was 6.36 seconds.

Discussion

Experiment 2 shows that 7-10 year old children can judge whether the order of a set of elements is determined, and shows that with the very simple procedure used there is no age effect. Sternberg's (1980) model predicts that children will be able to recognize indeterminacy, but does not provide a megacognitive basis for explaining this ability. Furthermore there does not appear to be any data confirming that children have this ability. Some doubt must have existed as to whether children of this age can recognize indeterminacy because of controversy as to whether they can recognize logical necessity in reasoning (Falmagne, Mawby & Pea, 1989; Halford, 1982, in press; Markovits, Schleifer & Fortier, 1989; Moshman & Franks, 1986; Osherson & Markman, 1975; Tunmer, Nesdale & Pratt, 1983). Experiment 2 shows that ability to recognize indeterminacy exists in middle childhood. This confirms another prediction of the model.

Model's recognition of indeterminacy

A run of the model on the problem forms used in Experiment 2 is shown in Appendix D. On early problems, the model uses relational strategies, which do not recognize indeterminacy. It builds system level strategies first for those problem forms on which relational level strategies produce errors. This leads to a mixed-strategy phase, in which some but not all indeterminate problems are detected. However feedback after the second premise causes the model to build system level strategies for all problems, and the model then discriminates reliably between determinate and indeterminate forms. *Construction of ordered sets*

One of the main predictions of the model concerns the manner in which orders are constructed progressively as premises are processed. Therefore Experiment 3 is designed to examine the orders that children produce successively as premises are presented. These can be compared with the orders predicted by the model. Children's ability to order sets of four elements following a single exposure to each premise does not appear to have been tested. Also, their ability to order three elements, including a nonadjacent relation has not been tested. Children's ability to recognize indeterminacy while carrying out the ordering task will also be assessed, in order to check the findings of Experiment 2.

To avoid conflict between these two tasks, the ordering task will be carried on progressively as premises are presented, using cardboard cutouts of the premise elements. Determinacy judgments will be made by having the computer generate an order and having participants say whether it is correct, and whether it is the only order possible. When participants' and computer's orders are different, participants will be asked whether both could be right, and if not which one is right. Thus determinacy judgments will be made by indicating whether one or more than one order can be correct.

Experiment 3

Method

Participants

Participants were from a primary school in a lower middle-class suburb of Brisbane, Australia. The 15 younger participants had a mean age of 8 years 4 months, (range 7-8 to 9-3). The 15 older participants had a mean age of 11 years 5 months (range 10-1 to 12-1).

Apparatus was as for experiment 2, plus 12 cardboard cutout runners, two each of the colors yellow, blue, red, green, purple and white.

Problems

There were four types of problems. Problems 1-6 were three-term adjacent-and-nonadjacent problems, and problems 7-12 were four-term adjacent-only problems, as shown in Table 2. The remaining types of problems are shown in Table 4. Problems 13 and 14 were three-term indeterminate and problems 15-18 were four-term indeterminate. The possible orders that result from the indeterminate problems are shown in Table 4.

Insert Table 4 about here

Problem presentation. The three premises for each problem were presented as pairs of colored runner-sprites 2 cm high in the top right-hand corner of the monitor screen. The colors of the sprites were yellow, blue, red, green, purple and white. The pairs were presented sequentially, with the first pair at the top, and each successive pair below. They remained in view throughout the problem. All runners faced left towards a finish flag, with the leading runner on the left. For each problem the computer randomly selected the colors to be used, and randomly assigned colors to ordinal positions of runners.

Procedure. After each premise was presented, participants chose cutout runners from a pool and arranged them in the order that they considered appropriate for the premises presented so far. When they had done this and signalled that they understood the premises, the experimenter pressed the key which caused the computer to proceed to the next pair. The participant then updated the set of cutout runners. After the participants had completed their order following the third premise, they were asked whether there was any other way the cutout runners could line up. This will be referred to as the first determinacy question. If the answer was yes, s/he was asked to create the alternative order. If the answer was no, and the problem was indeterminate, the participant was queried about one of the runners whose position was indeterminate, e.g. "what about green, could he go somewhere else?" This is referred to as the second determinacy question. If the participant agreed that this runner could go in another position, s/he was asked to indicate the alternate position.

Practice problems consisted of an initial 4-term determinate problem (problem 7, Table 2) followed by three three-term determinate, three four-term determinate, and three four-term indeterminate problems, in random order. Complete feedback was given for practice problems; participants were shown any alternative orders they had not recognized, and any errors of ordering were corrected, making reference to the appropriate premises in both cases.

The 18 test problems were shown in random order. Then there were two catch problems, both of which were four-term determinate, but when participants had created their final order of runners, they were queried as to whether one of them could be placed in an alternate position, as with the second determinacy question.

Results and Discussion

In order to analyze participants' ability to recognize whether problems were determinate or indeterminate, a 2(grade) by 2(three term/four term) by 2(determinate/indeterminate) analysis of variance was performed on the proportion of problems (arcsine transformed) on which participants said the ordering was indeterminate on the first determinacy question. This produced a significant main effect of determinate/indeterminate, $F(1,28) = 49.29$, $p < .001$, and the interaction of age with determinate/indeterminate was marginally significant, $F(1,28) = 4.03$, $p = .06$. The (untransformed) mean proportion of problems said to be indeterminate for 7-9 year-olds was .12 for determinates and .44 for indeterminates. For 10-12 year-olds the proportions were .07 for determinates and .65 for indeterminates. Simple main effects analysis showed a significant determinate/indeterminate difference at both age levels, $F(1,28) = 12.60$ for the younger group and 40.70 for the older group. Therefore, both age groups showed significant discrimination between determinate and indeterminate problems.

It is possible however that the first indeterminacy question underestimates the accuracy of children's judgments in absolute terms. Therefore the same analysis was performed on the proportion of problems on which a correct determinacy judgment was given on either the first or second indeterminacy question. This yielded only one significant effect, an interaction of agexdeterminacyxthree-term versus four-term, $F(1,28) = 5.80$, $p < .05$. The means are shown in Figure 12. Consistent with Experiment 2, there is a high proportion of correct determinacy judgments for both age-groups, although the 7-9 year olds are relatively poorer on the three-term indeterminates, but even here their performance is quite high in absolute terms. It is possible that determinacy is more difficult to check on three-term problems because of the nonadjacent relation (a>c). This often means that three relations (a>b, b>c, a>c) must be processed to check whether the order of a set of three elements is determinate. On the catch questions (i.e. second determinacy question asked where the problem was determinate) an alternative order was suggested in only .10 of cases, and there were no age or problem form effects. This suggests that only a small proportion of answers to the first or second indeterminacy question are false alarms, confirming that children were genuinely able to recognize determinacy.

An analysis was also conducted on the proportion of the participants' orderings correct (i.e. consistent with the premises in indeterminate problems) after the three premises were presented, using the same factors as before. There was a main effect of three-term versus four-term, F(1,28) = 7.85, $p < .01$, and of determinate/indeterminate, F(1,28) = 25.56, $p < .001$. There was also an interaction of these two factors, F(1,28) = 20.54, $p < .001$. For the determinates the proportion correct was .96 for the three-term problems and .87 for the four-term problems. For the indeterminates the proportions were .97 and .98 respectively. This interaction is due to a low proportion correct for the determinate four-term problems. However there was a marginally significant interaction of these two factors with age, F(1,28) = 3.97, p = .06,indicating that the low proportion correct for these problems was primarily due to the fact that for the 7-9 year old participants the mean on these problems was .83 whereas for the 10-12 year old participants it was .90. Therefore the low performance on the determinate four-term problems was primarily due to the 7-9 year old children.

In 94 percent of cases where participants correctly recognized that a problem was indeterminate, they were able to construct both the alternate orders that were consistent with the premises. There were no age or problem form effects.

To summarise the results of experiment 3, both 7-9 and 10-12 year olds can discriminate significantly between determinate and indeterminate N-term series problems, without the benefit of an additional prompt. However with benefit of a prompt that asks whether a specific element could go in another position, their accuracy becomes quite high in absolute terms, and this is not due to false alarms. They can also find the correct order of elements in both four-term adjacent-only and three-term adjacent-and-nonadjacent problems with high accuracy.

Model's performance on problems in Experiment 3

A run of the model on problem forms in Experiment 3 is shown in Appendix E. On the first three problems, the model is operating at the relational level (effort < 1.30). By problem 4, the model is operating at the system level, and indeterminacies in the resultset are recognized. From this point on, the model can reproduce the output of the participants in Experiment 3. That is, it can construct an order consistent with the premises, recognize indeterminacies, and produce the alternate order. ***Experiments 1-3*** show that children can map one ordered set into another, that they can recognize determinacy in an ordered set, and that they can construct orders of elements in accordance with the premises, all consistent with the model. A further tenet of the model is that reasoning skills, as used in N-term series problems, are gradually constructed through experience in problem solving, and based on previous, less domain-specific knowledge. This suggests that cognitive development cognitive development is not to be explained in terms of a shift from less appropriate to more appropriate strategies. In our model children have no N-term series strategies to begin with. They have a concept of order and manipulative knowledge that have been acquired in past experience, and with these they gradually acquire problem solving skills.

One way to check this is to attempt to find strategies for N-term series reasoning. If we find that young children have well-established strategies that are incorrect, and which explain their early failures, gradually being replaced by correct strategies, this would disconfirm the present model. Therefore Experiments 4 and 5 will attempt to analyse strategies used on problems of the types we have been assessing, using a response pattern analysis (sometimes known as a rule assessment) technique similar to that of Levine (1966) or Siegler (1981).

Children's Strategies in N-term Series Problems

Experiment 4

Method

Participants. There were 15 participants with a mean age of nine years 2 months (range 8-6 to 9-6) and 15 with a mean age of 12 years four months (range 11-8 to 13-6) both from a primary school in a lower-middle class suburb of Brisbane, Australia.

Apparatus was as for Experiment 3, plus a piece of cardboard with a line and a flag on it representing the finish-line.

Problem presentation was based on colored runner-sprites as for Experiment 3. In addition there was a judge-sprite with a cup trophy in the bottom-left corner of the screen, symbolizing the finish of a hypothetical race. The participant indicated that s/he had comprehended the most recent premise by lining up the cardboard cutout runners, after which the experimenter pressed a key which caused the computer to display copies of the runner-sprites in a line-up before the judge. If premises after the first introduced new runners, these were added to the lineup. If runners in the computer's lineup had to be reordered this was done by having runners hurdle one another. Sometimes the computer's line-up was consistent with the premises, and sometimes inconsistent.

Procedure. After each premise was presented, participants chose cutout runners from a pool and arranged them in the order that they considered appropriate for the premises presented so far. When they had done this, signalling that they understood the premises, the experimenter pressed the key which caused the computer to update the lineup at the judge in the bottom-left corner of the screen. The participant was then asked to compare his/her ordering of the cutouts with the ordering produced by the computer. If they were the same, the participant was asked if there was any other way the runners could line up consistent with the premises. If participant's and computer's orders were different the participant was asked to say which order was correct, or whether either might be correct. No feedback was given for the judgments on test problems because feedback might induce a change of strategy during the problem set, thereby violating the assumptions of the method, that strategies must remain constant over a set of problems used to make a diagnosis (Levine, 1966).

Problem forms. There were seven problem forms, comprising the six possible forms of the three-term adjacent-and-nonadjacent problem, and the four-term nonadjacent problem cd,ab,bc, as shown in Table 5. These problem forms were found to provide the best discrimination between possible strategies, as shown in different patterns of responses. The order of premises within each set was the order in which the pairs of runner-sprites were presented in the top-right corner of the screen. After each premise was presented the computer lined up copies of the runner-sprites at the judge in the bottom-left corner of the screen. These computer-generated orders are shown in Table 5. For the first two problem forms, the computer generated three different orders in front of the judge (problems 1-3, 4-6 in Table 5), and it generated two orders for each of the remaining five problems forms (problems 7-16). Problems 1-14 were presented once each, and problems 15 and 16 were presented three times, making a total of 20 problems. Each participant was presented with the complete set of problems, in random order. The color of runner-sprite was randomly allocated to each element in a problem, with the restriction that the same color could not occur twice in a problem.

Insert Table(s) 5 about here

Practice problems. The first six problems consisted of a single premise. The computer generated a pair of runners in the top-right corner, and a copy at the judge in the bottom-left corner. On half the trials the order of runners at the judge was consistent with the premise, and on half the trials it was inconsistent. Participants were asked to say whether the order of runners at the judge was consistent with the premises. The purpose was to ensure that participants were willing to accept or reject the computer-generated ordering, as appropriate. The next eight practice problems were the same except that two premises were presented. The computer-generated order was always consistent with the first premise after the first premise had been shown (first pair stage). Following the second premise (second pair stage), the computer-generated order was inconsistent with the first premise on 25 percent of trials and with the second premise on 25 percent of trials. Participants were encouraged to use the cardboard cutout runners as described for the test problems. *Strategy diagnosis*

There were 7 strategies that were both identifiable on logical grounds and for which evidence was obtained either in previous research (Maybery et al., in preparation) or in pilot studies. These strategies may be partitioned first into those that were based on an integration of the information in all three premises, and those that were based on only part of the premise information. There was a single strategy in the first category, shown as the integration strategy in Table 5. This strategy can be illustrated with the premise sets ac,ab,bc and cd,ab,bc. With the former premise set, it would be recognized that the order was indeterminate after the first two premises, so the alternative orders acb and abc would be stored, or one of these orders would be stored with a marker (as suggested by Foos et al., 1976) between b and c indicating that the relation between them was unknown. With the second premise set, the two premises cd,ab would be first stored separately, or would be stored as a string with a marker between d and a indicating that their order was unknown. Then when bc was presented, the order would be adjusted to take account of both the most recent premise and the stored information. That is, ab would be switched as a pair to the front of the string, giving abcd. This takes account of all the premise information. A strategy which integrates information from all premises requires that indeterminacies be recognized. It depends on constructing partial orders, and resolving the indeterminacies when further premises are presented.

The remaining strategies take account of only part of the premise information. The first subcategory contains strategies based on using one of the end terms as an anchor. In the front strategy, the term at the front of the string was identified if it occurred first in both the first two premises. For example in the a>b,b>c problem in Table 5, the first two premises are ab,bc, so no term occurs first, and the order is deemed to be indeterminate. However in the first a>b,a>c problem, the first two premises are ab,ac and in this case a occurs first in both premises, so it is used as the front element. Any order produced by the computer that has a as the first element will be accepted. The back strategy was similar except that it was based on the end term, identified as an element that was last in the first two premises.

The remaining strategies focussed on the most recent premise. The first of these was the adjacent, described in the introduction. It entails placing the elements of the last premise in the correct order adjacent to each other. Thus in the a>b,b>c problem in Table 5 the orders ab,abc,acb are produced after the first, second and third premises respectively. The incorrect order acb results from placing c next to a following the last premise, ac. The remote strategy also places elements in the order indicated by the last premise, but places them as far apart as possible. Thus in the first a>b,a>c problem in Table 5, the first two premises, ab,ac produce the order abc, but the third premise, bc results in b being shifted to the front, yielding bac.

The adjacent-confirming strategy is similar to the adjacent strategy but is modified so that items already in position, that are consistent with the current premises, are left as they are. In the a>b,b>c problem, the order abc results from the first two premises, ab,bc, then when ac is presented, this is consistent with the existing order abc, so it is left alone. This avoids the error made in the adjacent strategy, of shifting c next to a. The sequential strategy focusses on the most recent premise, but the placement of elements is influenced by the order in which they appear in the premises. In the first b>c,a>c problem the first two premises are bc,ac. The sequential strategy places c last, consistent with the premises, but since the order of b and a is indeterminate, they are placed in the order they appear, yielding bac. The indeterminacy after the first two premises is not recognized, as it would be in the integration strategy.

Table 5 shows the pattern of participants' responses that would be expected for each strategy. There is one pattern at the second pair stage and one at the third pair stage. A response was scored accept (a) if the participant constructed the same order as the computer, and said this was the only order possible, or if they constructed a different order and said the computer's order was correct. A response was scored reject (r) if the participant constructed a different order from the computer, and said his/her order was the correct one. Responses were scored indeterminate (i) if the participant constructed the same order as the computer but said another order was possible, or if they constructed a different order from the computer and said either order might be correct.

Insert Table(s) 7 about here

Consider, for example, problem 1. After the premises ab,bc have been presented, the computer generates the order abc. This is consistent with the order that would have been generated by the integration strategy. Therefore a participant using this strategy would accept this order, so an (a) is shown for the second premise stage of problem 1 for the integration strategy. When the third premise, ac is presented, the computer generates the order acb. This is inconsistent with the order generated by the integration strategy, so a participant using that strategy would reject that order, and accordingly an (r) is shown for the third premise stage of problem under the integration strategy. On the other hand a participant using the adjacent strategy would accept the computer's order for problem 1 at both the second and third premise stage, so an (a) is shown for both stages under this strategy. In problem 7, the order is indeterminate under the integration strategy at the second premise stage, so (i) is shown at this stage for this strategy. The minimum number of problems on which strategies differed in the responses predicted was six, summed over second and third premise stages.

Strategy diagnosis was based primarily on acceptance of the computer's order because this technique has been shown to be a sensitive measure of strategies (Briars and Siegler, 1984), and because it covers cases where children might make one of two or more orders, any of which they would be prepared to accept as valid. If there is more than one order consistent with a strategy, the child would be prepared to accept any of them, thereby avoiding false negatives.

The accept, reject or indeterminate response made by each participant at the second premise stage, for the full set of problems, was pattern-matched against the pattern shown in Table 5 for each strategy. This process was repeated for the third premise stage.

Results

Normally a participant provided 40 judgments, summed over the second- and third-premise stages of the 20 problems. A participant was diagnosed as having used a strategy if at least 80 percent of his/her judgments matched the pattern for that strategy, and if no other strategy produced an equal-best match. One of the older participants completed only 17 useable trials, and one of the older and one of the younger participants completed only 19 useable trials, due to experimental error. In this case the criterion was 80 percent of available judgments. All the rest completed 20 trials. Where a participant's judgments matched the patterns for two strategies equally, this was resolved by allocation to the strategy that best matched the orders constructed by the participant. This had to be done for three participants, and in each case the match on the orders was better than 80 percent. The number of participants using each strategy is shown in Table 6. Significantly more older- than younger- participants were diagnosed as having used a specific strategy, X2(1) = 4.60, p < .05.

Insert Table(s) 6 about here

The figures in parentheses in Table 6 show how many participants could be considered to have used a strategy if the criterion is their best-matching strategy, irrespective of the percentage of trials that match, with equal matches being resolved as before. The mean percentage of trials that matched the best-fitting strategy for participants who were diagnosed by this criterion, but who did not meet the 80 percent criterion, was 70.1 percent, range 60-77.5 percent. By either criterion, the majority of participants who used a diagnosable strategy used the integration strategy.

The data were examined for evidence of strategies that had not been envisaged in the design. One strategy was found, the switching strategy, which was a modification of the adjacent strategy, but entailed moving both the elements mentioned in a premise. Ten occurrences were observed for the 8-9 year olds, and 14 for the 5-7 year olds, as shown in Table 9. However no participant used it consistently enough to meet the 80 percent criterion.

In order to analyse the correctness of participants' final order with the cutouts, the 20 problems were divided into three categories. There were six problems in the first category (problems one to six), eight in the second (problems seven to 14), and because 15 and 16 were each presented 3 times, there were 6 problems in the last category. Problems 1-6 are alike in having three terms and requiring a>b,b>c or b>c,a>b processes as defined by Foos et al. (1976) and problems 7-14 were alike in having three terms and producing indeterminacies at the second premise stage. Problems 1-14 were alike in using nonadjacent relations. Problems 15-16 were alike in having four terms, and using only adjacent relations.

A two (age) x three (problem type) ANOVA was conducted on the proportion (arc-sine transformed) of final cutout orders correct. This yielded a significant effect of age, F(1, 28) = 4.37, p < .05, and of problem type, F(2, 56) = 27.60, p < .001. The mean proportion correct for the eight- to nine-year olds was .73, and for the 11- to 13-year olds was .83. The mean proportion correct for problems one- to six was .83, for problems seven- to fourteen was .96, and for problems 15-16 was .54. The four-term adjacent-and-nonadjacent problems are more difficult, probably because the two separate strings cd,ab have to be held in STM and then integrated with the third premise bc.

Experiment 5.

The purpose of Experiment 5 was to examine N-term series processes used by 5-7 year old children, using the same method as in Experiment 4.

Participants

There were 15 children, 7 girls and 8 boys, from a campus after-school daycare centre. Their mean age was 6 years, five months, range 5-8 to 7-5. There were an additional two participants who failed to reach the practice criterion, specified below, and one who was color-blind.

Apparatus and procedure were the same as experiment 4, except that the practice-problem set was expanded so there were 16 two-premise practice problems instead of 8. Half the problems were determinate and half were indeterminate. The criterion for practice was one determinate problem where the participant's order was correct although the computer's order was incorrect, and at least one indeterminate problem on which the participant produced an order consistent with the premises. Participants were also required to recognize whether a premise was consistent with his/her order, and with the computer's order. Participants were also required to achieve three consecutive correct problems at some point in practice. At least 8 practice problems were given.

Results

The scoring criteria were the same as for Experiment 4, and the results are shown in Table 6. Only 40 percent of participants met the 80 percent criterion. As with Experiment 4, the most common single strategy was integration, but there were no significant strategy preferences.

A one-way ANOVA was performed on proportion (arc-sine transformed) of orders correct, with the same three levels of the problem-form factor used in Experiment 4. This yielded F(2, 28) = 31.56, p < .001. The mean proportion correct was .97 for a>b,b>c and b>c,a>b problems, .91 for a>b,a>c and b>c,a>c problems, and .31 for four-term problems. Again, four-term problems were most difficult.

Error analysis

Problems 15-16 were the most difficult. This would be expected because they require two premises to be held in STM before they can be integrated. Therefore it was thought worthwhile to examine the error patterns on these problems in more detail, and this was done for the pooled data from Experiments 4 and 5. The most common error was to produce an ordering that was consistent with the adjacent strategy (i.e. bcda or dabc). The proportion (arc-sine transformed) of errors on problems 15-16 that were consistent with the adjacent strategy was subjected to a one-way Analysis of Variance with three levels of age, but there was no significant effect. The mean (untransformed) proportion was .67, which differs significantly from the chance value of .09, $t(37) = 9.11$, $p < .01$. (Note the dfs are reduced because some participants made no errors on these problems). Therefore the errors that were made were consistent with the use of the adjacent strategy, but it was not used consistently across all problems, being evident on the most difficult problems.

Model's performance on problems in Experiments 4 and 5

A trace of the models output for problem forms used in Experiments 4 and 5 is shown in Appendix F. The run began with previously built relational and system level productions in place. The production strength threshold was set at the low value of .15, so the weaker relational productions had a chance to appear. The level of production, relational (rel) or system (sys) is shown in Appendix F. Under "resultset" are shown the order which the model constructed in the course of its normal operation, and the order suggested in accordinance with the methodology of Experiments 3 and 4. Where sysem level productions are fired, the model tends to construct an order consistent with the premises, and correctly deals with the suggested order by accept, reject, or indeterimate judgments.

In the instances of problem form 11 which occur in Appendix F, the indeterminacy marker was lost from the resultset due to random memory failure, as suggested in the work of Foos et al., (1976), discussed earlier. This causes errors, mainly on those problem forms which entail a high memory load. This is true of problem form 11 (Table 2), with premises c>d, a>b, b>c, because a string of four elements, cdab, plus the indeterminacy marker, must be held in short term memory after processing the second premise. This simulates the periodic errors made by both children and adult participants on this problem form.

Conclusions from Experiments 4 and 5.

Two strategy-diagnosis experiments have shown that no single strategy appears to give an adequate account of N-term series performance by children aged 5-13 years. Quite a high proportion of children appeared to use no diagnozable strategy with 80 percent consistency, and this was more true for younger participants. Interpretation of this finding should be made in recognition of the possibility that there may have been inaccuracies of diagnosis. Nevertheless close inspection of the strategies by independent observers did not disclose any consistent pattern in undiagnosed performances. Another alternative is that some participants, particularly the younger ones, did not use a single strategy, but changed their procedure from time to time.

The evidence is consistent with the theory that N-term series reasoning processes are gradually acquired piecemeal, on a problem-by-problem basis. Incorrect performances are not caused by incorrect strategies, in the sense of generalized procedures that apply to all problems, but simply reflect inadequately developed skills. These skills are acquired gradually, as experience with a variety of problem forms accumulates.

General Discussion

This paper has presented a computer simulation of the development of a problem solving skill in a particular domain, N-term series reasoning or transitive inference. However the principles incorporated in this model are likely to apply to other domains, and to that extent it might be regarded as a general theory of the development of reasoning skills.

The immediate aims of the model were to account for previous data on N-term series reasoning, and to predict some hitherto unobserved aspects of the process. In respect of the first aim, it has been shown that the model can account for the way people acquire the ability to construct an ordered set in working memory. This process has been central to previous models of N-term series reasoning (Foos et al., 1976; Sternberg, 1980; Trabasso, 1977). What the present model adds is an account of how this ability is acquired. The specific strategies or skills required to construct an ordered set consistent with the premises are not assumed, but are developed as the system gains experience with the task.

Another aspect of previous work on N-term series is the difficulty of integrating premises. There is evidence that young children, and adults when unfamiliar with the task or under high processing loads, tend to process premises singly, without taking sufficient account of previous premises (Baylor & Gascon, 1974; Halford, 1984; Halford & Kelly, 1984). More sophisticated, or more competent performers integrate premises, finding a solution that is consistent not only with the most recent premise but with previous ones as well. It has also been shown that premise integration is associated with high processing loads (Maybery et al., 1986; Halford et al., 1986). All these premise integration phenomena have been incorporated into the model.

Another source of failure in N-term series reasoning is the tendency to encode premises in absolute rather than relational terms (Breslow et al., ; Perner & Mansbridge, 1983; Siegler, 1989; Sternberg & Rifkin, 1979). Simulation of premise encoding would require a natural language processing model and is beyond the scope of the present project. However the type of encoding might well be a consequence, rather than a cause, of difficulty in integrating premises. The reason is that if premises are processed separately there is no need to avoid absolute labels. For example, if we say Bill is taller than John, no conflict results from coding Bill as tall and John as short. Conflict does occur however if we consider two premises jointly. Consider Bill is taller than John and John is taller than Mike. The problem now is that John is coded as both tall and short. Absolute coding therefore causes conflict when premises are considered jointly. If participants are unable to process premises jointly because of processing loads, but are restricted to processing them serially (one-at-a-time) there is less incentive to avoid absolute encoding. Thus absolute encoding could be consequence of information processing demands. If however premises are processed jointly a coding scheme must be adopted which avoids conflict; e.g. large, medium, small, or top, middle, bottom. Thus coding might be a consequence of the amount of information that processed in a single decision.

The present model predicts that young children should have the ability to map one ordered set into another, because this structure mapping ability is a core feature of the model. Experiment 1 shows that most 5-year old children do this. The specific ability to map ordered sets does not appear to have been assessed previously, but it should be noted that a mapping of this form is an example of what Gentner (1983) calls systematicity, because it is based on a coherent set of relations. However Gentner & Toupin (1986) found that 5-7 year olds were not able to make structure mappings based on systematicity. The observation that 5-year olds can do so is therefore both new and important to the theory of N-term series reasoning.

We have also shown that children can recognize whether the order of a set of elements is determinate. Recognition of indeterminacy is postulated in the model of Sternberg (1981), but children's ability to perform this task has not been tested previously. Doubt as to whether they can do so is caused by controversy as to whether they can recognize that a logical argument (based on two categorical premises) is determinate (see Halford 1989 for a review). Experiments 2 and 3 show that they can do so, consistent with the model. This does not necessarily imply that the can recognize when a logical argument is determined, or can recognize logical necessity, because the tasks are too different to permit extrapolation from one to the other.

The study has also shown that strategies are unlikely to be the explanation for performance on N-term series tasks. Experiments 4 and 5 failed to find evidence that failures on N-term series reasoning were attributable to incorrect strategies. Consistent with this finding, the model does not use strategies as the cause of behaviour, but treats strategies and skills as phenomena that need to be explained. It is consistent in this respect with the planning net models of Van Lehn and Brown (1980) and Greeno et al. (1984). A planning net is a directed graph in which the nodes are plans or strategies, and the links between the nodes are constraints exercized by the task concept on the strategies. Planning nets have been used by Greeno et al. (1984) to develop a model of how a child's concept of number constrains counting strategies. Our model shows how a concept of order can constrain the development of N-term series reasoning skills.

However our model differs from previous planning net models in two significant ways. The first is that it permits concrete experience with specific content to provide the declarative knowledge base for development of reasoning skills. That is, experience with specific ordered sets can be used as the basis for a concept of order, which in turn can constrain the development of N-term series reasoning skills. By contrast Greeno et al. (1984) postulate knowledge of universally valid logical rules, which are probably innate. We do not object to postulating the existence of such knowledge, but believe that a role must be provided for empirical knowledge as well, because not all reasoning skills can be based on innate knowledge. Furthermore logical reasoning principles appear to have limited power to explain human reasoning (Halford, 1990).

Use of content-specific, experience-based knowledge to constrain acquisition of reasoning skills is made possible in our model by the use of analogical mapping. This is the second way that our model differs from previous planning net models. Analogies have the advantage that they permit concrete instances of concepts to be used as a basis for inferences that go beyond those instances. Analogies are a form of abstraction, as was pointed out in the introduction. They make it unnecessary to postulate knowledge of abstract, universally valid, logical rules or principles, because specific instantiations of those principles can suffice. In our model a representation of an ordered set of at least three elements serves as a concept of order, and permits children to reason as though they understood the abstract concept of order.

The third way that our model differs from previous planning net models is that it does not postulate that understanding operates throughout all performances of a cognitive task. Previous planning net models have modelled the role of understanding in the sense that they have shown how a concept of the task constrains the development of strategies. The present theory entails a role for understanding in this way, but it also postulates that once skills, modelled as production rules, are acquired, understanding is no longer required. When a production exists for a particular task it simply fires, equivalent to automatically performing a task we already know how do. Understanding is only invoked when it is necessary to acquire a new problem solving skill.

The model also blends metacognitive and associative strategy selection mechanisms. Planning net models are essentially metacognitive models, in that they model the way strategies and skills are developed under the constraint of declarative knowledge. However as Siegler's (1989) work has shown, a great deal of strategy growth and development can be account for by associative mechanisms. In the present model strategies are strengthened or weakened, once they are acquired, by associative learning mechanism. Furthermore associative mechanisms are the first resort, because performance is based on existing skills where they are applicable, and development proceeds by strengthening or weakening the processing underlying these skills. It is only if existing skills are not applicable that resort is had to metacognitive mechanisms.

The model also posulates that processing loads play a role in skill acquisition. The specific mechanism postulated in this paper entails using a previously experienced ordered set as an analog of the order which must be constructed to perform the task. As explained in the Introduction, this entails structure mapping, which inmposes a processing load. The general principle is that analogical reasoning imposes a high processing load when more than one relation (or large amounts of structural information) must be processed in a single mapping decision. A lot of N-term series reasoning entails mapping two or more relations into an ordering schema, and the joint processing of two relations imposes a high processing load.

This processing load is separate from memory loads that might apply during other aspects of task performance. It is distinct from the short term memory load imposed when an ordered set is stored in working memory during processing of the premises. The short term storage load applies whenever information has to be stored for later use, and is distinct from processing load, which is imposed by information that is actually being processed. This distinction is considered in more detail, together with supporting evidence elsewhere (Halford, in press, Chapter 3). Structure mapping, which occurs when reasoning skills are being developed, imposes a processing load. This load does not occur once skills are acquired.

Transitive inference has been associated with a history of failure in young children, the causes of which have been the subject of much controversy (Halford, 1989). It has been proposed that there are a number of sources of false negatives in the tests that have been used, but Halford (in pressa,b) has argued that these factors, while important, do not completely explain the difficulties experienced, especially the rather persistent problems exhibited by young children. Furthermore Halford et al., (1986) and Maybery et al. (1986) have shown that the processing load associated with premise integration is a source of difficulty for both children and adults.

The STAR model of analogical reasoning outlined earlier provides a new account of these processing loads. It implies that because transitivity is a 3-dimensional concept (based on a ternary relation), representation of the transitivity principle requires a rank-4 tensor product, which is computationally costly. This produces a tendency to default to lower dimensional representations, in which only one binary relation can be represented at a time.

The present model implies that one of the things that develops is the ability to represent integrated relations. Young children, as well as older children and adults under lower effort or high conflicting loads, tend to represent only one relation. This can be compared to Siegler's (1981) principle that children first represent a variety of tasks, including proportion, balance scale, and conservation, in terms of a single, dominant dimension, and only later integrate this with the second, subordinate dimension. For example, in the balance scale, early representations tend to be based on weight, which is progressively integrated with distance.

The progression observed in this research from representations based on one relation to those which integrate two relations, and Siegler's finding of progression from representation of a single, dominant dimension to integration of dominant and subordinate dimensions, can be seen as two cases of a common principle. Both imply that development entails *progression to representations of higher dimensionality*. This appears to be emerging as one of the fundamentally important principles of cognitive development.

Transitive inference has been associated with Piaget's (1950) concrete operational stage of cognitive development. The present model, unlike Piaget's, is not based on psycho-logic, but emphasizes learning and reasoning mechanisms more in keeping with the theory of Anderson (1983, 1987). Nevertheless it would explain why transitive inference has been found to be a difficult task for young children. The high processing load imposed by the requirement to map premises into an ordering schema would place an especially heavy burden on young children. Note that the basic mechanisms are the same at all ages, and the processing load is the same for any participant at a given level of expertise. It appears however that young children cannot construct representatins of sufficiently high dimensionality to understand transitivity. As Halford (in press) has suggested, the observations that gave rise to Piaget's stage theory might reflect the development of representations of higher dimensionality, and tensor products of higher rank, with age.

The model is generally consistent with Anderson's (1983) ACT* model, and with Anderson's (1987) theory of skill acquisition. It postulates the use of domain-general methods such as analogical reasoning and means-end analysis to construct skills, based on previously acquired declarative knowledge. In its present form it does not incorporate the compilation and proceduralization postulated by Anderson (1987). These processes could be used to explain gains in efficiency as a result of further experience with the task. However the aim of the present project has been to simulate the acquisition of basic reasoning skills in the domain of N-term series. It also had the aim of providing a sufficiency test for some basic developmental mechanisms, which will be considered in the next section.

Developmental significance of the model

A fundamental problem for cognitive development is to explain how children acquire cognitive processes that are autonomous and adaptive, so the child can deal with situations and problems that could not have been anticipated. This implies the ability to develop strategies which meet task demands as they arise. It is not sufficient to explain children's performance by describing the strategies they use, because the really important problem is to account for the development of the strategies. To account for children's transitive inferences by defining their strategies is undoubtedly useful, especially if the strategy is defined precisely, and is rigorously validated. Some earlier models of this kind formed an important foundation for this project. But to treat the strategy as the explanation for the performance is tantamount to assuming that the strategy somehow preexisted, specifically for performing that task. It is like assuming that child has a program for performing transitive inferences in the laboratory. Actually however there is no reason why a child should be equipped to perform transitive inference tasks in a laboratory. Such performances are of interest only because they reflect the autonomous, adaptive processes by which a child manages to cope with a multitude of problems in real life. Research on these performances is useful to the extent that it tells us about these adaptive processes.

We began therefore with the assumption that there is no reason why children or adults should have preexisting strategies for performing laboratory-based transitive inferences. When asked to do so, they build strategies that draw on past knowledge, and which meet the unanticipated demands of the task. That knowledge was not acquired with the purpose of using it in this way. It might have been acquired for a quite different purpose, or might have been acquired through relatively aimless exploration of the world. The problem then was to explain how knowledge of the world, acquired for a different purpose or without any purpose, can be adapted to meet the demands of an unanticipated problem. This seems to us to be more a "real-life" situation than most experimental paradigms.

The task demanded that children take a set of relational premises, and make certain inferences that follow from the set, but which cannot be made from any premise taken alone. This entails integrating the premises into an ordered set. The knowledge required to do this includes the concept of an ordered set. It was necessary therefore to explain the nature and origin of this concept. For reasons given earlier, it is unlikely that children have a formal mathematical definition of an ordered set, or even domain-general principles that relate to it.

The solution which seemed plausible to us was to assume that children acquire experience with specific, concrete examples of ordered sets, through play and general interaction with their world. Those experiences would be stored in memory, not for the purpose of solving N-term series problems, but simply as part of the child's world knowledge. The next problem was to explain how such knowledge could be used to guide the development of reasoning strategies.

Analogical reasoning proved very useful. Analogies are by no means new in models of skill acquisition. For example, Anderson's (1987) model incorporates analogies between the current problem solving step and a step taken in similar circumstances, in a related context, previously. However our use of analogy differs in that it enables knowledge acquired in a completely unrelated context to be used to guide the building of strategies for an unanticipated task. We believe this feature of our model captures an important aspect of the autonomous, adaptive processes that are fundamental to cognitive development.

It is also this step which imposes the processing load. Learning, in the sense of building up a store of representations of the world, does not appear to impose a significant processing load. We appear to be well equipped with efficient mechanisms which detect regularities in the environment, and mental models of such regularities can be built up in small increments, without apparent effort. Processes by which this occur have been discussed elsewhere (Halford, in press; Holland et al., 1986). Significant processing loads are observed when it is necessary to map such experiences into new problems. The size of the loads depends on the complexity of structure which must be mapped. According to the STAR model of Halford et al. (in press) the load occurs because of the type of representation that is required for these structures. The important point here is that it is the need to map one structure into another that imposes the load.

This means that very young children might well have examples of ordered sets stored in memory, and might be able to learn N-term series strategies, if appropriately taught. What they evidently find difficult is to map their stored experiences into the tasks demanded of them. Thus they find it difficult to develop strategies for structurally complex tasks autonomously, based on their own representation of the task. To use an old fashioned word, they find it difficult to develop strategies with *understanding*.

There are other aspects of autonomous, adaptive behavior which are also captured by the model. One is that children should recognize when strategies are inappropriate. A clear case is that they should not be applied mindlessly to N-term problems which are unsolvable. Our data show children can do this, and the model shows that strategies based on knowledge of the world can reproduce this aspect of adaptive cognition. Thus the model is adaptive and autonomous in this sense also. Therefore we suggest that the computational model provides a sufficiency test for a theory of processes underlying adaptive, autonomous strategy and skill acquisition.

REFERENCES

Anderson, J.R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J.R. (1987). Skill acquisition: compilation of weak-method problem solutions. *Psychological Review, 94*, 192-210.

Baddeley, A.D. (1986). *Working Memory.* Oxford: Clarendon Press.

Baddeley, A.D. (1990). *Human memory: Theory and practice.* Needham Heights, MA: Allyn and Bacon.

Bakker, P.E., & Halford, G.S. (1988). *A basic computational theory of structure-mapping in analogy and transitive inference.* Unpublished Tech. Rep. 88/1, Centre for Human Information Processing and Problem Solving, University of Queensland, Brisbane, Australia.

Baylor, G.W., & Gascon, J. (1974). An information processing theory of aspects of the development of weight seriation in children. *Cognitive Psychology*, 1-40.

Brainerd, C.J., & Kingma, J. (1984). Do children have to remember to reason: A fuzzy-trace theory of transitivity development. *Developmental Review, 4*, 311-377.

Breslow, L. (1981). Reevaluation of the literature on the development of transitive inferences. *Psychological Bulletin, 89*, 325-351.

Briars, D., Siegler, R.S. (1984). A featural analysis of preschoolers' counting knowledge. *Developmental Psychology, 20*, 607-618.

Brown, A.L. (1989). Analogical learning and transfer: what develops? In S. Vosniadou & A. Ortony (Eds.), *Similarity and anlogical reasoning* (pp. 369-412). Cambridge: Cambridge University Press.

Bryant, P. (1989). Commentary on Halford (1989). *Human Development, 32*(6), 369-374.

Bryant, P.E., & Trabasso, T. (1971). Transitive inferences and memory in young children. *Nature, 232*, 456-458.

Bullock, M., & Gelman, R. (1977). Numerical reasoning in young children: The ordering principle. *Child Development, 48*, 427-434.

Chi, M.H.T., & Ceci, S.J. (1987). Content knowledge: its role, representation and restructuring in memory development. *Advances in Child Development and Behaviour, 20*, 91-142.

DeLoache, J.S. (1989). Young children's understanding of the correspondence between a scale model and a larger space. *Child Development, 4*, 121-139.

Falkenhainer, B., Forbus, K.D. & Gentner, D. (1990). The structure-mapping engine: algorithm and examples. *Artificial Intelligence, 41*, 1-63.

Falmagne, R.J., Mawby, R.A. & Pea, R.D. (1989). Linguistic and logical factors in recognition of indeterminacy. *Cognitive Development*, *4*(2), 141-176.

Foos, P.W., Smith, K.H., Sabol, M.A., & Mynatt, B.T. (1976). Constructive processes in simple linear order problems. *Journal of Experimental Psychology: Human Learning and Memory*, *2*, 759-766.

Gelman, R., & Gallistel, C.R. (1978). *The child's understanding of number*. Cambridge, MA: Harvard University Press.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155-170.

Gentner, D., & Toupin, C. (1986). Systematicity and surface similarity in the development of analogy. *Cognitive Science*, *10*, 277-300.

Goswami, U. (in press). Analogical reasoning: What develops? A review of research and theory. *Child Development*.

Greeno, J. G., & Johnson, W. (1984). Competence for solving and understanding problems. Paper presented at the International Congress of Psychology, Acapulco.

Greeno, J.G., Riley, M.S., & Gelman, R. (1984). Conceptual competence and children's counting. *Cognitive Psychology*, *16*, 94-143.

Halford, G.S. (1978). Cognitive development stages emerging from levels of learning. *International Journal of Behavioural Development*, *1*, 341-354 .

Halford, G.S. (1980). A learning set approach to multiple classification: Evidence for a theory of cognitive levels. *International Journal of Behavioural Development*, *3*, 409-422.

Halford, G.S. (1982). *The development of thought*. Hillsdale, NJ: Erlbaum.

Halford, G.S. (1984). Can young children integrate premises in transitivity and serial order tasks? *Cognitive Psychology*, *16*, 65-93.

Halford, G.S. (1987). A structure-mapping approach to cognitive development. *International Journal of Psychology* , *22*, 609-642.

Halford, G.S. (1989). Reflections on 25 years of Piagetian cognitive developmental psychology, 1963-1988. *Human Development*, *32*, 325-387.

Halford, G.S. (1990). Is children's reasoning logical or analogical? Further comments on Piagetian cognitive developmental psychology. *Human Development*, *33*, 356-361.

Halford, G.S. (in press). *Children's understanding: the development of mental models*. Hillsdale, N.J.: Erlbaum.

Halford, G.S., & Kelly, M.E. (1984). On the basis of early transitivity judgements. *Journal of Experimental Child Psychology*, *38*, 42-63.

Halford, G.S. & Leitch, E. (1989). Processing load constraints: a structure-mapping approach. In M.A. Luszcz & T. Nettelbeck (Eds.), *Psychological development:*

perspectives across the life-span (pp. 151-159). : Elsevier.

Halford, G.S., Maybery, M.T., & Bain, J.D. (1986). Capacity limitations in children's reasoning: A dual task approach. *Child Development, 57*, 616-627.

Halford, G.S., & Wilson, W.H. (1980). A category theory approach to cognitive development. *Cognitive Psychology, 12*, 356-411.

Halford, G.S., Wilson, W.H., Guo, J., Wiles, J. and Stewart, J.E.M. (in press). Connectionist implications for processing capacity limitations in analogies. In K. J. Holyoak & J. Barnden (Eds.), *Advances in Connnectionist and Neural Computation Theory, Vol. 2: Analogical Connections* . Norwood, NJ: Ablex.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., & Thagard, P.R. (1986). *Induction: Processes of inference, learning and discovery.* Cambridge, MA: Bradford Books/MIT Press.

Holyoak, K. (1991). Symbolic connectionism: Towards third-generation theories of expertise. In Ericsson, K.A. & Smith, J. (Eds.), *Toward a general theory of expertise : prospects and limits* . Cambridge, U.K.: Cambridge University Press.

Holyoak, K.J., & Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory and Cognition, 15*, 332-340.

Holyoak, K.J., & Thagard, P. (1989). Analogical mapping by constraint satisfactions. *Cognitive Science, 13*(3), 295-355.

Humphreys, M.S., Bain, J.D., & Pike, R. (1989). Different ways to cue a coherent memory system: A theory for episodic, semantic and procedural tasks. *Psychological Review, 96*(2), 208-233.

Kahneman, D. (1973). *Attention and effort.* Englewood Cliffs, NJ: Prentice-Hall.

Kallio, K.D. (1982). Developmental change on a five-term transitive inference. *Journal of Experimental Child Psychology, 33*, 142-164.

Levine, M. (1966). Hypothesis behaviour by humans during discrimination learning. *Journal of Experimental Psychology, 71*, 331-338.

Markovits, H., Schleifer, M., & Fortier, L. (1989). Development of elementary deductive reasoning in young children. *Developmental Psychology, 25*(5), 787-793.

Maybery, M.T. (1987). *Information processing models of transitive inference.* Unpublished Ph.D. Thesis, University of Queensland, Brisbane, Australia.

Maybery, M.T., Bain, J.D., & Halford, G.S. (1986). Information processing demands of transitive inference. *Journal of Experimental Psychology: Learning, Memory and Cognition, 12*, 600-613.

Maybery, M.T., Halford, G.S., Bain, J.D., & Kelly, M.E. (in preparation). *Children's construction of series.* Unpublished paper, University of Western Australia, Nedlands, WA.

Moshman, D., & Franks, B.A. (1986). Development of the concept of inferential validity. *Child Development, 57*, 153-165.

Ohlsson, S., & Langley, P. (1986). *PRISM tutorial and manual.* Irvine, CA: University of California.

Osherson, D.N., & Markman, E. (1975). Language and the ability to evaluate contradictions and tautologies. *Cognition, 3*(3), 213-216.

Pears, R. and Bryant, P. (1990). Transitive inferences by young children about spatial position. *British Journal of Psychology, 81*(4), 497-510.

Perner, J., & Mansbridge, D.G. (1983). Developmental differences in encoding length series. *Child Development, 54*, 710-719.

Piaget, J. (1950). *The psychology of intelligence.* (M. Piercy & D.E. Berlyne, Trans.) London: Routledge & Kegan Paul, (Original work published 1947).

Piaget, J. (1954). *The construction of reality in the child.* (M. Cook, Trans.) New York: Basic Books, (Original work published 1950).

Rumelhart, D.E., & McClelland, J.L. (Eds.) (1986). *Parallel distibuted processing: Explorations in the microstructure of cognition.* Cambridge, MA: MIT Press.

Rumelhart, D.E., Smolensky, P., McClelland, J.L. & Hinton, G.E. (1986). Schemata and sequential thought processes in PDP models. In McClelland, J.L. & Rumelhart, D.E. (Eds.), *Parallel distibuted processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models* (pp. 7-57). Cambridge, MA: MIT Press.

Schneider, W. & Detweiler, M. (1987). A connectionist/control architecture for working memory. *The Psychology of Learning and Motivation, 21*, 53-119.

Siegler, R.S. (1981). Developmental sequences within and between concepts. *Monographs of the Society for Research in Child Development, 46*, 1-84.

Siegler, R.S. (1987). Some general conclusions about children's strategy choice procedures. *International Journal of Psychology, 22*, 729-749.

Siegler, R.S. (1989). How domain-general and domain-specific knowledge interact to produce strategy choices. *Merrill-Palmer Quarterly, 35*(1), 1-26.

Siegler, R.S., & Jenkins, E.A. (1989). *How children discover new strategies.* Hillsdale, NJ: Lawrence Erlbaum.

Siegler, R.S., & Shrager, J. (1984). Strategy choices in addition and subtraction: How do children know what to do? In C. Sophian (Ed.), *Origins of cognitive skills* (pp. 229-293). Hillsdale,NJ: Erlbaum.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence, 46*(1-2), 159-216.

Sternberg, R.J. (1980a). The development of linear syllogistic reasoning. *Journal of Experimental Child Psychology, 29*, 340-356.

Sternberg, R.J. (1980b). Representation and process in linear syllogistic reasoning. *Journal of Experimental Psychology: General, 109*, 119-159.

Sternberg, R.J. (1981). Reasoning with determinate and indeterminate linear syllogisms. *British Journal of Psychology*, *72*, 407-420.

Sternberg, R.J., & Rifkin, B. (1979). The development of analogical reasoning. *Journal of Experimental Child Psychology*, *27*, 195-232.

Thayer, E.S., & Collyer, C.E. (1978). The development of transitive inference: A review of recent approaches. *Psychological Bulletin*, *85*, 1327-1343.

Touretzky, D.S. & Hinton, G.E. (1988). A distributed connectionist production system. *Cognitive Science*, *12*, 423-466.

Trabasso, T. (1975). Representation, memory, and reasoning: How do we make transitive inferences? In A.D. Pick (Ed.), *Minnesota symposia on child psychology* (Vol. 9, pp. 135-172). Minneapolis: University of Minnesota Press.

Trabasso, T. (1977). The role of memory as a system in making transitive inferences. In R.V. Kail, Jr. & J.W. Hagen, *Perspectives on the development of memory and cognition* (pp.333-366). Hillsdale, NJ: Erlbaum.

Tunmer, W.E., Nesdale, A.R., & Pratt, C. (1983). The development of young children's awareness of logical inconsistencies. *Journal of Experimental Child Psychology*, *36*, 97-108.

VanLehn, K., & Brown, J.S. (1980). Planning nets: A representation for formalizing analogies and semantic models of procedural skills. In R.E. Snow, P.A. Federico, & W.E. Montague (Eds.), *Aptitude learning and instruction. Vol. 2. Cognitive process analyses of learning and problem solving* (pp. 95-137). Hillsdale, NJ: Erlbaum.

Appendix A

Productions existing before the model begins operation.

```
;          Productions     - for getting premises
;                          - for reporting on solution states
;                          - for retrieving info from ltm


; If there is nothing else to do, do something.

(READY
 (
  (FLAG (wm waiting))
  (<not> (FLAG (get feedback)))
 )
 -->
 (
  ($clear-dm wm)
  ($add-to wm (GOAL (do-something)) 1 1 1)
 )
)
0.4

;if the goal is to do something, read in more input

(GET_NEW_INPUT
 (
  (GOAL (do-something))
 )
 -->
 (
  ($add-to wm (INPUT (($input-line))) 1 1 0)
  ($delete-from wm (GOAL (do-something)))
 )
)
0.3

; If the new input is recognised as a command, accept it
; as a goal to fulfill.
; Valid commands:- add, restart, stop, report, say, other.

(NEXT_COMMAND
 (
  (INPUT (=new-input))
  ((@ltm (recognized-commands !f =new-input !b))
 )
 -->
 (
  ($add-to wm (GOAL (=new-input)) 1 1 1)
  ($delete-from wm (INPUT (=new-input)))
 )
)

; Get next premise interactively and set goal to integrate-premise.

(ASK_FOR_PREMISE
 (
  (GOAL (=reply) )
  (*or(*equal =reply add)(*equal =reply ADD) )
 )
 -->
 (
  ($writecr Enter the premise: )
  ($add-to wm
      (premise (($input-line))) 1 1 0
      (GOAL (integrate-premise)) 1 1 1
  )
  ($delete-from wm (GOAL (=reply)))
 )
```

```
)

; If result set is empty add first premise directly.

(INTEGRATE_FIRST_PREMISE
 (
 (GOAL (integrate-premise))
 (<not> (result set (=rel =rset)))
 (premise (=ob1 =rel =ob2))
 )
 -->
  (
  ($delete-from wm
      (GOAL (integrate-premise))
       (premise (=ob1 =rel =ob2))
   )
   ($add-to wm
       (GOAL (do-something)) 1 1 1
        (result set (=rel (=ob1 =ob2))) 1 1 0
   )
   ($trace-results (=ob1 =rel =ob2) (=ob1 =ob2))
  )
)

; If result set is not empty add a goal to search for
; premise elements in resultset.

(INTEGRATE_OTHER_PREMISES
 (
 (GOAL (integrate-premise) )
 (<not> (MATCHLIST !matchlist))
 (<not> (FLAG (result-set-modified)));23-3-91
 (result set (=rel =resultset) )
 )
 -->
 (
 ($add-to wm (result set (=rel =resultset)) 1 1 0 )
 ($add-to matchmem
     (GOAL (make MATCHLIST)) 1 1 1
     (MATCHSET =resultset) 1 1 0
 )
 ($call mm)
 )
)

(TRY_EXISTING_STRATEGIES

 ;check new-pm for existing strategies,
 ;if there are none,
 ;production STRATEGIES_TRIED sets the goal to build a new production.

 (
  (GOAL (integrate-premise))
  (MATCHLIST =list1 =list2)
  (result set (=rel =resultset))
  (<not> (FLAG (STRUCTURE MAPPER called)))
 )
 -->
 (
  ($call pm new-pm )
 )
)

; This production allows the model to continue though
; no strategy already exists.

(STRATEGIES_TRIED
 (
    (MATCHLIST =list1 =list2)
    (<not> (GOAL (build a production)))
 )
 -->
 (
 ($add-to wm (GOAL (build a production)) 1 1 1)
 ($call pm)
 )
)
0.5

; Production which fires in pm after new-pm strategy is applied.
; This prevents another premise integration cycle beginning.
; It would fail as some conditions are removed by new-pm strategy productions.

(STRATEGY_APPLIED
 (
 (FLAG (result-set-modified))
 (result set (=rel =resultset))
 (premise =premise)
 )
 -->
 (
 ($trace-results =premise =resultset)
 ($delete-from wm
      (FLAG (result-set-modified))
      (GOAL (integrate-premise))
      (GOAL (build a production))
      (premise =premise)
 )
 ($add-to wm (GOAL (do-something)) 1 1 1)
 )
)

; Once premise is integrated, set goals to look for
; fresh input.

(READY_FOR_NEW_PREMISE
 (
 (<not> (premise =premise))
 (result set (=rel =resultset))
 )
 -->
 (
 ($delete-from wm (GOAL (integrate-premise)))
 ($add-to wm (GOAL (do-something)) 1 1 1)
 )
)

; If trying to achieve a goal, and there is nothing else
; to do, find a concept that is linked with that goal in ltm.

(MAP_GOAL_TO_CONCEPT
 (
 (GOAL (build a production))
 (<not> (FLAG (STRUCTURE MAPPER called)))
 (<not> (FLAG (result-set-modified)))
 (@ltm
          (=name
           !headlist
           (strategy !head (=goal (map-concept =concept) !rest) !tail)
           !taillist)
 )
 )
 -->
 (
 ($add-to wm (GOAL (check =concept)) 1 1 1)
 )
)

; If wanting to check the current state of wm against a concept of order,
; call the Structure Mapper, giving it the current premise and resultset.

(CALL_STRUCTURE_MAPPER
(
 (GOAL (check concept-of-order))
 (premise =premise)
 (result set (=rel =resultset))
 (<not> (FLAG (STRUCTURE MAPPER called)))
 )
 -->
 (
 ($add-to wm
      (FLAG (STRUCTURE MAPPER called)) 1 1 1
      (result set (=rel =resultset)) 1 1 0
     (premise =premise) 1 1 0
  )
 ($delete-from wm
      (result set (=rel =resultset))
     (premise =premise)
     (GOAL (check concept-of-order))
  )
 ($consider sm)
 ($call  sm)
 )
)

; Elements are added to the matcher memory so that a feature-list
; will be built.

(MATCH_ON_SMRESULT
 (
  (smresult set =smresult)
  (FLAG (STRUCTURE MAPPER called))
  (premise (=ob1 =rel =ob2) )
  (<not> (GOAL (find-ltm-match)))
  (<not> (SM_MATCHLIST !list1))
 )
 -->
 (
  ($delete-from wm (smresult set =smresult))
  ($add-to matchmem
      (GOAL (make SM_MATCHLIST)) 1 1 1
      (GOAL (find-ltm-match)) 1 1 1
      (MATCHSET =smresult) 1 1 0
  )
  ($call mm)
 )
)

(BUILD_A_NEW_BODY
 (
  (GOAL (build a production))
  (NEWPERFORM =perform)
  (MATCHLIST (=ob1 !rest1) (=ob2 !rest2))
 )
 -->
 (
  ($delete-from wm (NEWPERFORM =perform))
  ($add-to wm (BODY
    (=name
     (
      (result set (/=rel /=resultset))
      (MATCHLIST (/=ob1 !rest1 ) (/=ob2 !rest2 ))
     )
     -->
     (
     (/$delete-from wm (MATCHLIST (/=ob1 !rest1) (/=ob2 !rest2)))
     (/$delete-from wm (result set (/=rel /=resultset)))
     (/$add-to wm (result set (/=rel
                   ((/$perform =perform /=ob1 /=resultset /=ob2 )))) 1 1 1)
    (/$writecr =perform performed)
     (/$add-to wm (FLAG (result-set-modified)) 1 1 1)
     )
    )
  ) 1 1 0)
 )
)

(BUILD_A_PRODUCTION
 (
  (GOAL (build a production))
  (BODY =body)
  (*not (*already-built =body new-pm))
  (SM_MATCHLIST (=ob1 =E =F)(=ob2 =G =H))
 )
 -->
 (
  ($build-in new-pm =body)
  ($delete-from wm
          (GOAL (build a production))
          (GOAL (do-something))
          (FLAG (STRUCTURE MAPPER called))
          (BODY =body)
          (SM_MATCHLIST (=ob1 =E =F)(=ob2 =G =H))
  )
 )
)

; Retrieve an operator from ltm to make transition from
; current to goal state.

(MATCH_AGAINST_LTM
 (
  (consider =a-number-of =relations)
  (MATCHLIST (=ob1 =A =B)(=ob2 =C =D))
  (SM_MATCHLIST (=ob1 =E =F)(=ob2 =G =H))
  (GOAL (find-ltm-match))
  (@ltm
   ((considering =a-number-of =relations)
                   ((=ltm-ob1 !front1 =A !mid1 =B !end1 )
                    (=ltm-ob2 !front2 =C !mid2 =D !end2 )
                    =perform
                    (=ltm-ob1 !front3 =E !mid3 =F !end3 )
                    (=ltm-ob2 !front4 =G !mid4 =H !end4 ))))
 )
 -->
 (
  ($add-to wm  (NEWPERFORM  =perform) 1 1 0)
  ($delete-from wm (GOAL (find-ltm-match)))
 )
)

; If the command is to stop, call the finishing prod's

(FINISH
 (
  (GOAL (=reply) )
  (*or(*equal =reply stop) (*equal =reply STOP) )
 )
 -->
 (
  ($end-trace)
  ($add-to wm
      (FLAG (halt)) 1 1 1
      (FLAG (wm waiting)) 1 1 1
      (FLAG (get feedback)) 1 1 1
      (=reply) 1 1 0
  )
 )
)

; Call an explicit halt.

(HALT
 (
 (FLAG (halt))
 )
-->
 (
 ($delete-from wm (FLAG (halt)) )
 ($halt)
 )
)

(RESTART_NEW_PROBLEM
 (
  (GOAL (=reply) )
 (*or(*equal =reply restart)(*equal =reply RESTART) )
 )
-->
 (
 ($trace-new-problem)
 ($add-to wm
      (FLAG (wm waiting) ) 1 1 1
     (GOAL (get-feedback)) 1 1 1
  )
 )
)

(FEEDBACK_COMMAND
 (
 (GOAL (get-feedback))
 (<not> (FLAG (no-feedback)))
 )
 -->
 (
 ($delete-from wm (GOAL (get-feedback)))
 ($add-to wm
          (FLAG (get feedback)) 1 1 1
          (GOAL (do-something)) 1 1 1
  )
 )
)
0.7

(GET_FEEDBACK
 (
  (INPUT (=reply))
  (result set (=rel =resultset) )
  (FLAG (get feedback))
 )
 -->
 (
  ($writecr)
  ($writecr "result set =" =resultset )
  ($writecr)
  ($writecr "enter the correct result eg. (a b c) or (indeterminate)")      ;(can't tell)")
  ($add-to wm
          (correctset is =reply) 1 1 0
          (GOAL (give feedback )) 1 1 1
          (GOAL (do-something )) 1 1 1
  )
  ($delete-from wm
          (FLAG (get feedback))
          (INPUT (=reply))
  )
 )
)
0.65

(GIVE_POS_FEEDBACK
 (
  (GOAL (give feedback))
  (result set (=rel =resultset) )
  (correctset is =correctset)
  (*equal =resultset =correctset)
```

BEST COPY AVAILABLE

```
 )
 -->
 (
  ($writecr "GIVING FEEDBACK TO LATEST ACTION")
  ($delete-from wm (GOAL (give feedback)))
  ($strength-feedback new-pm + 0.1)
  ($writecr "STRENGTH FEEDBACK DONE")
  ($writecr)
  ($update-effort + )
  ($writecr "EFFORT UPDATE DONE")
  ($writecr)
  ($delete-from wm (correctset is =correctset))
 )
)
0.65

(INDET_RESULT_POS_FEEDBACK
 (
  (GOAL (give feedback))
  (result set (=rel (!f << !m1 ^ !m2 >> !b)) )
  (correctset is (indeterminate))
 )
 -->
 (
  ($writecr "GIVING FEEDBACK TO LATEST ACTION")
  ($delete-from wm (GOAL (give feedback)))
  ($strength-feedback new-pm + 0.1)
  ($writecr "STRENGTH FEEDBACK DONE")
  ($writecr)
  ($update-effort + )
  ($writecr "EFFORT UPDATE DONE")
  ($writecr)
  ($delete-from wm (correctset is (indeterminate)))
 )
)
0.65

(GIVE_NEG_FEEDBACK
 (
  (GOAL (give feedback))
  (result set (=rel =resultset) )
  (correctset is =correctset)
  (*not (*equal =resultset =correctset))
 )
 -->
 (
  ($writecr "GIVING FEEDBACK TO LATEST ACTION")
  ($delete-from wm (GOAL (give feedback)))
  ($strength-feedback new-pm - 0.25)
  ($writecr "STRENGTH FEEDBACK DONE")
  ($writecr)
  ($update-effort - )
  ($writecr "EFFORT UPDATE DONE")
  ($writecr)
  ($delete-from wm (correctset is =correctset))
 )
)
0.65

; A series of productions for coping with exceptions.

; If no ltm match is found, raise effort and relax strategy
; selection threshold to increase likelihood of a match in future.

(LTM_MATCH_FAILURE
 (
  (MATCHLIST (=ob1 =A =B)(=ob2 =C =D))
  (SM_MATCHLIST (=ob1 =E =F)(=ob2 =G =H))
  (GOAL (find-ltm-match))
 )
 -->
 (
  ($up-effort)
  ($alter-new-pm-selection-threshold -0.05)
  ($delete-from wm
          (GOAL (find-ltm-match))
          (FLAG (STRUCTURE MAPPER called))
          (BODY =body)
          (GOAL (build a production))
   )
  ($add-to wm (bad-choice =new-sym) 1 0 1
   )
 )
)

; Two competing processes are working here. On one hand effort is increased
; so that if the model is close to a strategy transition threshold, it has the
; opportunity to cross.
; On the other, the criterion for considering unsuccessful productions is
; relaxed so that in the absence of fitter strategies, some existing strategy
; may operate.
; A third point is that there is a cut-off applied so that if the model is
; completely unable to solve a problem it escapes by abandoning it.

(BAD_CHOICE
 (
  (BODY =body)
  (*already-built =body new-pm)
 )
 -->
 (
  ($up-effort)
  ($alter-new-pm-selection-threshold -0.05)
  ($print-new-pm-selection-threshold)
  ($delete-from wm
       (FLAG (STRUCTURE MAPPER called))
       (BODY =body)
   )
  ($add-to wm (bad-choice =new-sym) 1 0 1)
 )
)

; If the current result set contains an indeterminacy,
; and the model has alreday failed to recover any appropriate
; operator, try ignoring the indeterminacy.

(ABANDON_INDET
 (
  (MATCHLIST !f (=ob =indic indet) !b)
  (result set (=rel =resultset))
  (bad-choice =c1)
  (bad-choice =c2)
  (*not (*equal (bad-choice =c1) (bad-choice =c2)))
 )
 -->
 (
  ($delete-from wm
     (MATCHLIST !f (=ob =indic indet) !b)
     (result set (=rel =resultset))
   )
  ($add-to wm
     (($lose-matchlist-indet (MATCHLIST !f (=ob =indic indet) !b))) 0.5 1 0
     (GOAL (find-ltm-match)) 1 1 1
     (result set (=rel ($remove-old-indeterminacy (=resultset)))) 1 1 0
   )
 )
)

(ABANDON_CURRENT_PROBLEM
 (
  (GOAL (integrate-premise))
  (MATCHLIST =list1 =list2)
  (bad-choice =c1)
  (bad-choice =c2)
  (bad-choice =c3)
  (*not (*equal (bad-choice =c1) (bad-choice =c2)))
  (*not (*equal (bad-choice =c1) (bad-choice =c3)))
  (*not (*equal (bad-choice =c2) (bad-choice =c3)))
  )
 -->
  (
  ($delete-from wm
    (bad-choice =c1)
    (bad-choice =c2)
    (bad-choice =c3)
     (main goal (=goal))
    (GOAL (integrate-premise))
     (MATCHLIST =list1 =list2)
   )
  ($add-to wm
    (FLAG (problem-abandoned)) 1 1 1
    (GOAL (do-something))      1 1 1
   )
  ($setq latest-action* nil)            ;ensure no feedback given in error
  )
 )

(LOOK_FOR_RESTART
 (
  (premise =arg)
  (FLAG (problem-abandoned))
  (INPUT (=reply) )
  (*or (*equal =reply restart)(*equal =reply RESTART)
       (*equal =reply stop)(*equal =reply STOP)
   )
  )
 -->
  (
  ($delete-from wm (FLAG (problem-abandoned)))
  ($add-to wm (FLAG (no-feedback)) 1 1 1)
  ($input-line)                                   ;throw away feedback info from input file
  )
 )

(IGNORE_OTHER_GOALS
 (
  (FLAG (problem-abandoned))
  (GOAL (=reply) )
  (@ltm (recognized-commands !f =reply !b))
  )
 -->
  (
  ($delete-from wm (GOAL (=reply)))
  ($add-to wm (GOAL (do-something)) 1 1 1)
  )
 )

; If a problem has already been abandoned as insoluble,
; ignore further information.

(IGNORE_INPUT
 (
  (INPUT (!new-input))
  (FLAG (problem-abandoned))
  )
 -->
  (
  ($delete-from wm (INPUT (!new-input)))
  ($add-to wm (GOAL (do-something)) 1 1 1 )
  )
 )

(RemoveSMFlag
 (
 (smresult set =smresult)
 (FLAG (STRUCTURE MAPPER called))
 )
 -->
 (
 ($delete-from wm (FLAG (STRUCTURE MAPPER called)))
 )
)
0.5

; Productions which output a judgement on the state of the result set.

(JUDGE
 (
 (GOAL (=reply))
 (*or (*equal =reply judge) (*equal =reply JUDGE))
 )
 -->
 (
 ($delete-from wm (GOAL (=reply)))
 ($add-to wm (GOAL (say-if-compatible)) 1 1 1)
 )
)

(ASK_FOR_ORDER
 (
 (GOAL (say-if-compatible))
 )
 -->
 (
 ($writecr possible order please\, in the form : (x y z) )
 ($add-to wm (correct? ($input-line)) 1 1 1)
 ($delete-from wm (GOAL (say-if-compatible)))
 )
)

(ACCEPT
 (
 (correct? =order)
 (result set (=rel =order))
 )
 -->
 (
 ($judgement-trace =order accepted)
 (writecr Order: !order is correct)
 ($add-to wm (GOAL (do-something)) 1 1 1)
 ($delete-from wm (correct? =order))
 )
)

(INDETERMINATE
 (
 (correct? =order)
 (result set (=rel =resultset))
 (*or (*equal =order (*perform remove_indet ob1 =resultset ob2))
      (*equal =order (*perform switch ob1 =resultset ob2)))
 )
 -->
 (
 ($judgement-trace =order order-indeterminate)
 ($dm-dump wm dmfile)
 (writecr Order: !order is indeterminate)
 ($add-to wm (GOAL (do-something)) 1 1 1)
 ($delete-from wm (correct? =order))
 )
)

(REJECT
 (
 (correct? =order)
```

BEST COPY AVAILABLE

```
 (result set (=rel =order2))
 (*not (*equal =order =order2))
 )
-->
 (
 ($judgement-trace =order rejected)
 (writecr Order: !order is incorrect)
 ($add-to wm (GOAL (do-something)) 1 1 1)
 ($delete-from wm (correct? =order))
 )
)

(OUTPUT_AN_ORDER
  (
   (GOAL (say-order))
   (result set (=rel =resultset))
   (*not (*member ^ =resultset))
  )
  -->
  (
   ($writecr order is =resultset)
   ($judgement-trace "order is " =resultset)
   ($delete-from wm (GOAL (say-order)))
   ($add-to wm (GOAL (do-something)) 1 1 1)
  )
)
0.4

(OUTPUT_ONLY_ORDER
  (
   (GOAL (other-order))
   (result set (=rel =resultset))
   (*not (*member ^ =resultset))
  )
  -->
  (
   ($judgement-trace "there is no other order" " " )
   ($delete-from wm (GOAL (other-order)))
   ($add-to wm (GOAL (do-something)) 1 1 1)
  )
)
0.4

(OUTPUT_BEST_INDET
  (
   (GOAL (say-order))
   (result set (=rel (!f ^ !b)))
  )
  -->
  (
   ($writecr order is ($perform remove_indet ob1 (!f ^ !b) ob2))
   ($judgement-trace "order is" (($perform remove_indet ob1 (!f ^ !b) ob2)))
   ($delete-from wm (GOAL (say-order)))
   ($add-to wm (GOAL (do-something)) 1 1 1)
  )
)

(OUTPUT_OTHER_INDET
  (
   (GOAL (other-order))
   (result set (=rel (!f ^ !b)))
  )
  -->
  (
   ($writecr order is ($perform switch ob1 (!f ^ !b) ob2))
   ($judgement-trace "order is " (($perform switch ob1 (!f ^ !b) ob2)))
   ($delete-from wm (GOAL (other-order)))
   ($add-to wm (GOAL (do-something)) 1 1 1)
  )
)
```

```
(SAY_ORDER
 (
  (GOAL (=reply))
  (*or (*equal =reply say) (*equal =reply SAY))
 )
 -->
 (
  ($add-to wm (GOAL (say-order)) 1 1 1)
  ($delete-from wm (GOAL (=reply)))
 )
)


(OTHER_ORDER
 (
  (GOAL (=reply))
  (*or (*equal =reply other) (*equal =reply OTHER))
 )
 -->
 (
  ($add-to wm (GOAL (other-order)) 1 1 1)
  ($delete-from wm (GOAL (=reply)))
 )
)

; report on the state of the problem.

(REPORT
  (
   (GOAL (=reply))
   (*or (*equal =reply report) (*equal =reply REPORT))
  )
  -->
  (
   ($add-to wm (GOAL (say-if-decidable)) 1 1 1)
  )
)

; Print a message if a problem is undecideable,
; i.e. the result set is indeterminate.

(UNDECIDABLE_PROBLEM
 (
  (GOAL (say-if-decidable))
  (result set (=relationx (!front << !ob1 ^ !ob2 >> !rest)))
 )
 -->
 (
  ($writecr The problem is undecidable. The order of !ob1 and !ob2 is unknown.)
  ($delete-from wm (GOAL (say-if-decidable)))
  ($add-to wm (GOAL (do-something)) 1 1 1)
 )
)

; Print a message if a problem has an unambiguous solution.

(DECIDABLE_PROBLEM
 (
  (GOAL (say-if-decidable))
  (result set (=relationx !resultset))
  (<not> (result set (=relationx (!front << !ob1 ^ !ob2 >> !rest))))
 )
 -->
 (
  ($writecr The order is determinate. The order is )
  ($write !resultset)
  ($delete-from wm (GOAL (say-if-decidable)))
  ($add-to wm (GOAL (do-something)) 1 1 1)
 )
)
```

```
);end build-in pm

; The following productions fire sequentially to build up a feature list for each object
; in the current premise, based on its position in the current result set.

; The two elements: (GOAL (make =matchlist)) and (MATCHSET !REST) are placed
; in matchmem by the productions, INTEGRATE_OTHER_PREMISES and
MATCH_ON_SMRESULT,
; which call the matching memory (mm).

; Indet is first noted, then removed before finding the other features.
; So the matchlist has the form:
;           ((ob1 absent|present|front|back unmarked|indet)
             (ob2 absent|present|front|back|in-front unmarked|indet))

(build-in mm


; If you have a goal to find the features associated with the
; objects in the current premise, as they occur in the result set,
; then tag each object so it can be fed through the matching productions.

(ASSERT_MATCH_ELEMENTS
 (
  (GOAL (make =matchlist))
  ((@wm (premise (=ob1 =rel =ob2)))
 )
-->
 (
  ($add-to  matchmem (=matchlist)
            matchmem (match =ob1)
            matchmem (match =ob2))
 )
)

(REMOVE_INDET

; If the MATCHSET contains an indeterminate part, then add it to matchmem
; as a new element and remove the indeterminacy markers from MATCHSET.
  (
   (GOAL (make =matchlist))
   (MATCHSET (!front << !m ^ !m2 >> !back))
  )
 -->
  (
  ($delete-from matchmem (MATCHSET (!front << !m ^ !m2 >> !back)))
  ($add-to     matchmem (MATCHSET (($remove-old-indeterminacy (!front << !m ^ !m2
>> !back))))
               matchmem (INDET (!m))
                  matchmem (INDET (!m2))
   )
  )
)

(MATCH_OBJ_INDET
  (
   (GOAL (make =matchlist))
   (match =ob)
   (match =other_ob)
   (=matchlist !f (=ob =pos) !b)
   (INDET (!front =ob !back))
   (*not (*member =other_ob ( !front !back)))
   (<not> (=matchlist !f (=ob =pos indet) !b))
  )
 -->
  (
  ($delete-from matchmem (=matchlist !f (=ob =pos) !b))
  ($add-to      matchmem (=matchlist !f (=ob =pos indet) !b))
  )
)
```

```
(MATCH_OBJ_UNMARKED
  (
   (GOAL (make =matchlist))
   (match =ob)
   (MATCHSET !REST)
   (=matchlist !f (=ob =pos) !b)
   (<not> (INDET (!front =ob !back)))
  )
  -->
  (
  ($delete-from matchmem (=matchlist !f (=ob =pos) !b))
  ($add-to      matchmem (=matchlist !f (=ob =pos unmarked) !b))
  )
)

(MATCH_OBJ_ABSENT
  (
   (GOAL (make =matchlist))
   (MATCHSET !REST)
   (=matchlist !list)
   (match =ob)
   (<not> (MATCHSET (!front =ob !back)))
   (<not> (=matchlist !f (=ob absent !other) !b))
   (<not> (MATCHSET (!front << !middle >> !back)))
  )
  -->
  (
  ($delete-from matchmem (=matchlist !list))
  ($add-to      matchmem (=matchlist !list (=ob absent )))
  )
)

(MATCH_OBJ_PRESENT
  (
   (GOAL (make =matchlist))
   (=matchlist !list)
   (match =ob)
   (MATCHSET (!front =ob !back))
   (<not> (=matchlist !f (=ob !other) !b))
   (<not> (MATCHSET (!front << !middle >> !back)))
  )
  -->
  (
  ($delete-from matchmem (=matchlist !list))
  ($add-to      matchmem (=matchlist !list (=ob present)))
  )
)

(MATCH_OBJ_FRONT
  (
   (GOAL (make =matchlist))
   (=matchlist !f (=ob =pos !other) !b)
   (<not> (=matchlist !f (=ob front !other) !b))
   (<not> (=matchlist !f (=ob in-front !other) !b))
   (match =ob)
   (MATCHSET (=ob !back))
   (<not> (MATCHSET (!front << !middle >> !back)))
  )
  -->
  (
  ($delete-from matchmem (=matchlist !f (=ob =pos !other) !b))
  ($add-to      matchmem (=matchlist !f (=ob front !other) !b))
  )
)

(MATCH_OBJ_BACK
  (
   (GOAL (make =matchlist))
   (=matchlist !f (=ob present !other) !b)
   (match =ob)
   (MATCHSET (!front =ob))
  (<not> (MATCHSET (!front << !middle >> !back)))
  )
 -->
  (
  ($delete-from matchmem (=matchlist !f (=ob present !other) !b))
  ($add-to      matchmem (=matchlist !f (=ob back !other) !b))
  )
)

(MATCH_OBJ_IN-FRONT
  (
  (GOAL (make =matchlist))
  ((@wm (premise (=ob1 =rel =ob2)))
  (MATCHSET (!front =ob2 !middle =ob1 !back))
  (=matchlist (=ob1 =pos1 =other1) (=ob2 =pos2 =other2 ))
  (*not (*equal =pos2 in-front))
  )
 -->
  (
  ($delete-from matchmem
                     (=matchlist (=ob1 =pos1 =other1) (=ob2 =pos2 =other2 )))
  ($add-to matchmem
                     (=matchlist (=ob1 =pos1 =other1) (=ob2 in-front =other2)))
  )
)

; Ensure that the objects are in the correct positions in the
; matchlist. If they are not, then switch them.

(ORDER_MATCHLIST
 (
 (GOAL (make =matchlist)) .
 ((@wm (premise (=ob1 =rel =ob2)))
 (=matchlist (=ob2 !pos2) (=ob1 !pos1))
 )
-->
 (
 ($delete-from matchmem (=matchlist (=ob2 !pos2) (=ob1 !pos1)))
 ($add-to      matchmem (=matchlist (=ob1 !pos1) (=ob2 !pos2)))
 )
)

(FINISHED_MATCHING
 (
 (GOAL (make =matchlist))
 (=matchlist (=ob1 !pos1) (=ob2 !pos2))
 )
 -->
 (
 ($add-to      wm (=matchlist (=ob1 !pos1) (=ob2 !pos2)) 1 1 1)
 ($clear-dm matchmem)
 ($call pm)                          ;return to main series productions
 )
)

);build-in mm
```

Appendix B

List of Productions Produced during run of model on 4 sets of problem forms 1 to 12.

```
(token-1 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 back unmarked) (=ob2 absent unmarked)))
        -->
       (($delete-from wm
                   (MATCHLIST (=ob1 back unmarked)
                            (=ob2 absent unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
               ($newadd
                (result set
                      (=rel (($perform append =ob1 =resultset =ob2))))))
        ($writecr append performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
1.6  2.775

(token-2 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 front unmarked) (=ob2 back unmarked)))
        -->
       (($delete-from wm
                   (MATCHLIST (=ob1 front unmarked)
                            (=ob2 back unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
               ($newadd
                (result set
                      (=rel
                      (($perform delete_append
                               =ob1
                               =resultset
                                =ob2))))))
        ($writecr delete_append performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.2725  0.75

(token-3 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 absent unmarked) (=ob2 front unmarked)))
        -->
       (($delete-from wm
                   (MATCHLIST (=ob1 absent unmarked)
                            (=ob2 front unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
               ($newadd
                (result set
                      (=rel (($perform insert =ob1 =resultset =ob2))))))
        ($writecr insert performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
1.44  2.575

(token-12 ((result set (=rel =resultset))
         (MATCHLIST (=ob1 front unmarked) (=ob2 absent unmarked)))
         -->
        (($delete-from wm
                    (MATCHLIST (=ob1 front unmarked)
                             (=ob2 absent unmarked)))
         ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                ($newadd
                 (result set
                       (=rel
                       (($perform append_indet
                                =ob1
                                =resultset
```

```
                          =ob2))))))
        ($writecr append_indet performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.826  1.8

(token-13 ((result set (=rel =resultset))
         (MATCHLIST (=ob1 back indet) (=ob2 in-front indet)))
        -->
        (($delete-from wm
                   (MATCHLIST (=ob1 back indet) (=ob2 in-front indet)))
         ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
               ($newadd
                (result set
                     (=rel
                      (($perform switch =ob1 =resultset =ob2))))))
        ($writecr switch performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.9975  1.8

(token-14 ((result set (=rel =resultset))
         (MATCHLIST (=ob1 absent unmarked) (=ob2 back unmarked)))
        -->
        (($delete-from wm
                   (MATCHLIST (=ob1 absent unmarked)
                          (=ob2 back unmarked)))
         ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
               ($newadd
                (result set
                     (=rel
                     (($perform insert_indet
                            =ob1
                            =resultset
                             =ob2))))))
        ($writecr insert_indet performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.7835  1.8

(token-15 ((result set (=rel =resultset))
         (MATCHLIST (=ob1 present indet) (=ob2 in-front indet)))
        -->
        (($delete-from wm
                   (MATCHLIST (=ob1 present indet)
                          (=ob2 in-front indet)))
         ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
               ($newadd
                (result set
                     (=rel
                      (($perform switch =ob1 =resultset =ob2))))))
        ($writecr switch performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.7417  1.4

(token-16 ((result set (=rel =resultset))
         (MATCHLIST (=ob1 present indet) (=ob2 back indet)))
        -->
        (($delete-from wm
                   (MATCHLIST (=ob1 present indet) (=ob2 back indet)))
         ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
               ($newadd
                (result set
                     (=rel
                     (($perform remove_indet
                            =ob1
                            =resultset
                             =ob2))))))
        ($writecr remove_indet performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
```

```
0.7185  1.4

(token-17 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 front indet) (=ob2 present indet)))
       -->
       (($delete-from wm
                 (MATCHLIST (=ob1 front indet) (=ob2 present indet)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
              ($newadd
               (result set
                    (=rel
                    (($perform remove_indet
                              =ob1
                              =resultset
                               =ob2))))))
       ($writecr remove_indet performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.7283  1.4

(token-18 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 absent unmarked) (=ob2 absent unmarked)))
       -->
       (($delete-from wm
                 (MATCHLIST (=ob1 absent unmarked)
                         (=ob2 absent unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
              ($newadd
               (result set
                    (=rel
                    (($perform concat_indet
                              =ob1
                              =resultset
                               =ob2))))))
       ($writecr concat_indet performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.8235  1.8

(token-19 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 present indet) (=ob2 present indet)))
       -->
       (($delete-from wm
                 (MATCHLIST (=ob1 present indet)
                         (=ob2 present indet)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
              ($newadd
               (result set
                    (=rel
                    (($perform remove_indet
                              =ob1
                              =resultset
                               =ob2))))))
       ($writecr remove_indet performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.7334  1.4

(token-20  ((result set (=rel =resultset))
        (MATCHLIST (=ob1 front unmarked) (=ob2 back unmarked)))
       -->
       (($delete-from wm
                 (MATCHLIST (=ob1 front unmarked)
                         (=ob2 back unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
              ($newadd
               (result set
                    (=rel
                    (($perform leave_alone
                              =ob1
```

```
                                 =resultset
                                  =ob2))))))
       ($writecr leave_alone performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.9019   1.6
```

Note. As new productions are built, names are assigned by the running model. In these appendices a consistent renaming has been applied so that the names always refer to the same production.

Appendix C

Sequence of Productions Fired in the Solution of Problem Forms 1, ,2, and 3.

```
READY
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_FIRST_PREMISE
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_OTHER_PREMISES
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_ABSENT
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
MAP_GOAL_TO_CONCEPT
CALL_STRUCTURE_MAPPER
MATCH_ON_SMRESULT
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
MATCH_AGAINST_LTM
BUILD_A_NEW_BODY
BUILD_A_PRODUCTION
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
token-1
STRATEGY_APPLIED
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_OTHER_PREMISES
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
MAP_GOAL_TO_CONCEPT
CALL_STRUCTURE_MAPPER
MATCH_ON_SMRESULT
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
MATCH_AGAINST_LTM
BUILD_A_NEW_BODY
BUILD_A_PRODUCTION
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
token-2
STRATEGY_APPLIED
GET_NEW_INPUT
NEXT_COMMAND
RESTART_NEW_PROBLEM
FEEDBACK_COMMAND
GET_NEW_INPUT
GET_FEEDBACK
GIVE_NEG_FEEDBACK
READY
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_FIRST_PREMISE
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_OTHER_PREMISES
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_ABSENT
MATCH_OBJ_PRESENT
MATCH_OBJ_UNMARKED
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
MAP_GOAL_TO_CONCEPT
CALL_STRUCTURE_MAPPER
MATCH_ON_SMRESULT
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
MATCH_AGAINST_LTM
BUILD_A_NEW_BODY
BUILD_A_PRODUCTION
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
token-3
STRATEGY_APPLIED
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_OTHER_PREMISES
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
MAP_GOAL_TO_CONCEPT
CALL_STRUCTURE_MAPPER
MATCH_ON_SMRESULT
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
MATCH_AGAINST_LTM
BUILD_A_NEW_BODY
BAD_CHOICE
TRY_EXISTING_STRATEGIES
MAP_GOAL_TO_CONCEPT
token-2
STRATEGY_APPLIED
GET_NEW_INPUT
NEXT_COMMAND
RESTART_NEW_PROBLEM
FEEDBACK_COMMAND
GET_NEW_INPUT
GET_FEEDBACK
GIVE_NEG_FEEDBACK
READY
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_FIRST_PREMISE
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_OTHER_PREMISES
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_ABSENT
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
MAP_GOAL_TO_CONCEPT
CALL_STRUCTURE_MAPPER
MATCH_ON_SMRESULT
REMOVE_INDET
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_INDET
MATCH_OBJ_FRONT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
FINISHED_MATCHING
MATCH_AGAINST_LTM
BUILD_A_NEW_BODY
BUILD_A_PRODUCTION
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
token-12
STRATEGY_APPLIED
GET_NEW_INPUT
NEXT_COMMAND
ASK_FOR_PREMISE
INTEGRATE_OTHER_PREMISES
REMOVE_INDET
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_INDET
MATCH_OBJ_PRESENT
MATCH_OBJ_INDET
MATCH_OBJ_IN-FRONT
MATCH_OBJ_BACK
FINISHED_MATCHING
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
MAP_GOAL_TO_CONCEPT
CALL_STRUCTURE_MAPPER
MATCH_ON_SMRESULT
ASSERT_MATCH_ELEMENTS
MATCH_OBJ_PRESENT
MATCH_OBJ_PRESENT
MATCH_OBJ_UNMARKED
MATCH_OBJ_BACK
MATCH_OBJ_UNMARKED
FINISHED_MATCHING
MATCH_AGAINST_LTM
BUILD_A_NEW_BODY
BUILD_A_PRODUCTION
TRY_EXISTING_STRATEGIES
STRATEGIES_TRIED
token-13
STRATEGY_APPLIED
GET_NEW_INPUT
NEXT_COMMAND
FINISH
HALT
```

BEST COPY AVAILABLE

Appendix D

Summary of results using the Problem Forms used in Experiment 2.

Premise	Resultset	Effort	Production fired
a > b	a b	0.6	
a > c	a c b	0.6	token-4
b > c	a b c	0.95	token-5
b > c	b c	0.94	
a > c	b a c	0.94	token-6
a > b	a b c	1.21	token-7
b > c	b c	1.2	
a > b	a b c	1.2	token-3
a > c	a c b	1.2	token-2
a > b	a b	1.4	
b > c	a b c	1.4	token-1
a > c	a b c	1.39	token-20
b > c	b c	1.38	
a > b	a b c	1.38	token-3
a > c	a b c	1.38	token-20
a > b	a b	1.37	
b > c	a b c	1.37	token-1
a > c	a b c	1.36	token-20
a > b	a b	1.36	
b > c	a b c	1.36	token-1
a > c	a b c	1.35	token-20
b > c	b c	1.34	
a > c	<< b ^ a >> c	1.34	token-14
a > b	a b c	1.34	token-15
b > c	b c	1.33	
a > c	<< b ^ a >> c	1.33	token-14
a > b	a b c	1.32	token-15
a > b	a b	1.32	
a > c	a << c ^ b >>	1.32	token-12
b > c	a b c	1.31	token-13
b > c	b c	1.3	
a > b	a b c	1.3	token-3
a > c	a b c	1.3	token-20
a > b	a b	1.29	
a > c	a << c ^ b >>	1.29	token-12
b > c	a b c	1.28	token-13
b > c	b c	1.28	
a > c	<< b ^ a >> c	1.28	token-14
a > b	a b c	1.27	token-15
b > c	b c	1.26	

```
a > b      a b c             1.26      token-3
a > c      a b c             1.26      token-20

a > b      a b               1.25
a > c      a << c ^ b >>     1.25      token-12
b > c      a b c             1.25      token-13

b > c      b c               1.24
a > b      a b c             1.24      token-3
a > c      a b c             1.23      token-20

a > b      a b               1.23
a > c      a << c ^ b >>     1.23      token-12
b > c      a b c             1.22      token-13

b > c      b c               1.21
a > c      << b ^ a >> c     1.21      token-14
a > b      a b c             1.21      token-15

a > b      a b               1.2
b > c      a b c             1.2       token-1
a > c      a b c             1.2       token-20

b > c      b c               1.19
a > c      << b ^ a >> c     1.19      token-14
a > b      a b c             1.18      token-15

a > b      a b               1.18
b > c      a b c             1.18      token-1
a > c      a b c             1.17      token-20

b > c      b c               1.17
a > b      a b c             1.17      token-3
a > c      a b c             1.16      token-20

a > b      a b               1.16
a > c      a << c ^ b >>     1.16      token-12
b > c      a b c             1.15      token-13

a > b      a b               1.14
b > c      a b c             1.14      token-1
a > c      a b c             1.14      token-20
```

Additional productions built in the course of this simulation.

```
(token-4 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 present unmarked) (=ob2 in-front unmarked)))
        -->
       (($delete-from wm
                  (MATCHLIST (=ob1 present unmarked)
                            (=ob2 in-front unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
              ($newadd
                (result set
                     (=rel
                     (($perform delete_insert
                              =ob1
                              =resultset
```

```
                                   =ob2))))))
        ($writecr delete_insert performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.513   1.1

(token-5 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 absent unmarked) (=ob2 front unmarked)))
         -->
        (($delete-from wm
                      (MATCHLIST (=ob1 absent unmarked)
                                 (=ob2 front unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                 ($newadd
                  (result set
                        (=rel (($perform insert =ob1 =resultset =ob2))))))
        ($writecr insert performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.877   1.6

(token-6 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 front unmarked) (=ob2 back unmarked)))
         -->
        (($delete-from wm
                      (MATCHLIST (=ob1 front unmarked)
                                 (=ob2 back unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                 ($newadd
                  (result set
                         (=rel
                         (($perform delete_append
                                   =ob1
                                   =resultset
                                   =ob2))))))
        ($writecr delete_append performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.391   0.875

(token-7 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 back unmarked) (=ob2 absent unmarked)))
         -->
        (($delete-from wm
                      (MATCHLIST (=ob1 back unmarked)
                                 (=ob2 absent unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                 ($newadd
                  (result set
                        (=rel (($perform append =ob1 =resultset =ob2))))))
        ($writecr append performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.835   1.6
```

Appendix E

Summary of Model's Performance on Problem Forms used in Experiment 3.

Problem Form Premise Resultset Effort Production fired

Problem Form	Premise	Result set	Effort	Production fired
7	a > b	a b	0.6	
	b > c	a b c	0.6	token-1
	c > d	a b c d	0.597	token-1
13	a > b	a b	0.594	
	a > x	a << x ^ b >>	0.594	token-12
	a > b	a << x ^ b >>	1.207	token-20
14	a > b	a b	1.201	
	x > b	a x b	1.201	token-6
	a > b	a x b	1.401	token-20
13	a > b	a b	1.551	
	a > x	a << x ^ b >>	1.551	token-12
	a > b	a x b	1.543	token-21
8	a > b	a b	1.657	
	c > d	<< a b ^ c d >>	1.657	token-18
	b > c	a b c d	1.649	token-19
9	b > c	b c	1.641	
	a > b	a b c	1.641	token-9
	c > d	a b c d	1.633	token-1
10		b > c	b c	1.624
	c > d	b c d	1.624	token-1
	a > b	a b c d	1.616	token-9
15		a > b	a b	1.608
	b > c	a b c	1.608	token-1
	x > b	<< a ^ x >> b c	1.7	token-23
16	b > c	b c	1.692	
	a > b	a b c	1.692	token-9
	b > x	a b << x ^ c >>	1.762	token-24
17	a > b	a b	1.754	
	a > x	a << x ^ b >>	1.754	token-12
	b > c	a x b c	1.856	token-1
16	b > c	b c	1.892	
	a > b	a b c	1.892	token-9
	b > x	a b << x ^ c >>	1.892	token-24
		order is	a b x c	
		order is	a b c x	
9	b > c	b c	1.919	
	a > b	a b c	1.919	token-9
	c > d	a b c d	1.919	token-1
		order is	a b c d	
		there is no other order		
7	a > b	a b	1.91	
	b > c	a b c	1.91	token-1
	c > d	a b c d	1.91	token-1
		order is	a b c d	
		there is no other order		
17	a > b	a b	1.9	
	a > x	a << x ^ b >>	1.9	token-12
	b > c	a x b c	1.944	token-1

```
                        order is                a x b c
                        there is no other order

1           a > b       a b                     1.958
            b > c       a b c                   1.958           token-1
            a > c       a b c                   1.958           token-20
                        order is                a b c
                        there is no other order

2           b > c       b c                     1.948
            a > b       a b c                   1.948           token-9
            a > c       a b c                   1.948           token-20
                        order is                a b c
                        there is no other order

5           a > c       a c                     1.938
            a > b       a << b ^ c >>           1.938           token-12
            b > c       a b c                   1.938           token-16
                        order is                a b c
                        there is no other order

13          a > b       a b                     1.929
            a > x       a << x ^ b >>           1.929           token-12
            a > b       a << x ^ b >>           1.929           token-21
                        order is                a x b
                        order is                a b x

6           a > c       a c                     1.919
            b > c       a b c                   1.919           token-6
            a > b       a b c                   1.919           token-22
                        order is                a b c
                        there is no other order

4           b > c       b c                     1.909
            a > c       b a c                   1.909           token-6

8           a > b       a b                     1.962
            c > d       << a b ^ c d >>         1.962           token-18
            b > c       a b c d                 1.962           token-19
                        order is                a b c d
                        there is no other order

15          a > b       a b                     1.952
            b > c       a b c                   1.952           token-1
            x > b       << a ^ x >> b c         1.952           token-23
                        order is                a x b c
                        order is                x a b c

14          a > b       a b                     1.942
            x > b       a x b                   1.942           token-6
            a > b       a x b                   1.942           token-20
                        order is                a x b
                        there is no other order

10          b > c       b c                     1.957
            c > d       b c d                   1.957           token-1
            a > b       a b c d                 1.957           token-9
                        order is                a b c d
                        there is no other order

11          c > d       c d                     1.947
            a > b       << c d ^ a b >>         1.947           token-18
            b > c       a b c d                 1.947           token-13
                        order is                a b c d
                        there is no other order

12          c > d       c d                     1.937
            b > c       b c d                   1.937           token-9
            a > b       a b c d                 1.937           token-9
                        order is                a b c d
                        there is no other order
```

3	a > b	a b	1.927	
	a > c	a << c ^ b >>	1.927	token-12
	b > c	a c b	1.927	token-13
		order is	a c b	
		there is no other order		
18	b > c	b c	1.946	
	x > c	b x c	1.946	token-6
	a > b	a b x c	1.946	token-9
		order is	a b x c	
		there is no other order		

The following are additional productions built in the course of solving the novel problem forms of Experiment 3 and hence not listed in Appendix B.

```
(token-21 ((result set (=rel =resultset))
         (MATCHLIST (=ob1 front unmarked) (=ob2 back indet)))
         -->
        (($delete-from wm
                     (MATCHLIST (=ob1 front unmarked) (=ob2 back indet)))
         ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                 ($newadd
                  (result set
                        (=rel
                         (($perform leave_alone =ob1 =resultset =ob2))))))
        ($writecr leave_alone performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.4502   0.975

(token-22 ((result set (=rel =resultset))
          (MATCHLIST (=ob1 front unmarked) (=ob2 present unmarked)))
          -->
         (($delete-from wm
                     (MATCHLIST (=ob1 front unmarked)
                               (=ob2 present unmarked)))
          ($delete-from wm (result set (=rel =resultset)))
          ($add-to wm
                  ($newadd
                   (result set
                         (=rel
                         (($perform leave_alone
                                   =ob1
                                   =resultset
                                    =ob2))))))
         ($writecr leave_alone performed)
          ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.5735   1.1

(token-23 ((result set (=rel =resultset))
          (MATCHLIST (=ob1 absent unmarked) (=ob2 present unmarked)))
          -->
         (($delete-from wm
                     (MATCHLIST (=ob1 absent unmarked)
                               (=ob2 present unmarked)))
          ($delete-from wm (result set (=rel =resultset)))
          ($add-to wm
                  ($newadd
                   (result set
                         (=rel
                         (($perform insert_indet
                                   =ob1
                                   =resultset
                                    =ob2))))))
         ($writecr insert_indet performed)
          ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.6117   1.2
```

```
(token-24 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 present unmarked) (=ob2 absent unmarked)))
        -->
       (($delete-from wm
                    (MATCHLIST (=ob1 present unmarked)
                             (=ob2 absent unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
               ($newadd
                (result set
                      (=rel
                      (($perform append_indet
                                =ob1
                                =resultset
                                 =ob2))))))
        ($writecr append_indet performed)
         ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.4123   0.975
```

Appendix F

Table of Results Summarizing the Model's Output for Problem Forms used in Experiments 4 and 5.

Form	Premise	Resultset	Effort	Rule Fired	Order	Judgement
1	a > b	a b	0.6			
	b > c	a b c	0.6	token-1	a b c	accepted
	a > c	a b c	0.6	token-20	a c b	rejected
6	a > c	a c	0.597			
	b > c	a b c	0.597	token-6	a b c	accepted
	a > b	a b c	0.597	token-9	a b c	accepted
2	b > c	b c	0.594			
	a > b	a b c	0.594	token-3	a b c	accepted
	a > c	a b c	0.594	token-20	b a c	rejected
4	b > c	b c	0.591			
	a > c	b a c	0.591	token-6	b a c	accepted
	a > b	a b c	0.591	token-7	a b c	accepted
11	c > d	c d	0.588			
	a > b	<< c d ^ a b >>	0.588	token-18	c d a b	indeterminate
	b > c	b c d a	1.206	token-5	a b c d	rejected
3	a > b	a b	1.404			
	a > c	a << c ^ b >>	1.404	token-12	a b c	indeterminate
	b > c	a b c	1.749	token-5	a b c	accepted
11	c > d	c d	1.74			
	a > b	<< c d ^ a b >>	1.74	token-18	c d a b	indeterminate
	b > c	b c d a	1.89	token-5	b c d a	accepted
3	a > b	a b	1.918			
	a > c	a << c ^ b >>	1.918	token-12	a c b	indeterminate
	b > c	a b c	1.965	token-5	b a c	rejected
11	c > d	c d	1.955			
	a > b	<< c d ^ a b >>	1.955	token-18	c d a b	indeterminate
	b > c	b c d a	1.955	token-5	b c d a	accepted
1	a > b	a b	1.967			
	b > c	a b c	1.967	token-1	a b c	accepted
	a > c	a b c	1.967	token-20	b a c	rejected
2	b > c	b c	1.957			
	a > b	a b c	1.957	token-3	a b c	accepted
	a > c	a b c	1.957	token-20	a c b	rejected
6	a > c	a c	1.947			
	b > c	a b c	1.947	token-6	b a c	rejected
	a > b	a b c	1.947	token-9	a c b	rejected
5	a > c	a c	1.937			
	a > b	a << b ^ c >>	1.937	token-12	a c b	indeterminate
	b > c	a b c	1.937	token-16	a b c	accepted
11	c > d	c d	1.928			
	a > b	<< c d ^ a b >>	1.928	token-18	c d a b	accepted
	b > c	b c d a	1.928	token-5	a b c d	rejected

BEST COPY AVAILABLE

4	b > c	b c	1.946			
	a > c	b a c	1.946	token-6	a b c	rejected
	a > b	a b c	1.946	token-7	a c b	rejected
11	c > d	c d	1.936			
	a > b	<< c d ^ a b >>	1.936	token-18	c d a b	indeterminate
	b > c	b c d a	1.936	token-5	b c d a	accepted
5	a > c	a c	1.952			
	a > b	a << b ^ c >>	1.952	token-12	a b c	indeterminate
	b > c	a b c	1.952	token-16	b a c	rejected
2	b > c	b c	1.942			
	a > b	a b c	1.942	token-3	a b c	accepted
	a > c	a b c	1.942	token-20	a b c	accepted
1	a > b	a b	1.932			
	b > c	a b c	1.932	token-1	a b c	accepted
	a > c	a b c	1.932	token-20	a b c	accepted
11	c > d	c d	1.923			
	a > b	<< c d ^ a b >>	1.923	token-18	c d a b	indeterminate
	b > c	<< b c d ^ a >>	1.967	token-5	a b c d	rejected

Additional productions built in the course of this simulation.

```
(token-8 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 present unmarked) (=ob2 back unmarked)))
         -->
        (($delete-from wm
                    (MATCHLIST (=ob1 present unmarked)
                            (=ob2 back unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                ($newadd
                  (result set
                        (=rel
                        (($perform delete_append
                                  =ob1
                                  =resultset
                                  =ob2))))))
        ($writecr delete_append performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.190   0.2

(token-9 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 front unmarked) (=ob2 present unmarked)))
         -->
        (($delete-from wm
                    (MATCHLIST (=ob1 front unmarked)
                            (=ob2 present unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
         ($add-to wm
                ($newadd
                  (result set
                        (=rel
                        (($perform delete_append
                                  =ob1
                                  =resultset
                                  =ob2))))))
        ($writecr delete_append performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.187   0.2
```

```
(token-10 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 absent unmarked) (=ob2 absent unmarked)))
        -->
        (($delete-from wm
                (MATCHLIST (=ob1 absent unmarked)
                        (=ob2 absent unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
            ($newadd
             (result set
                 (=rel
                 (($perform concat =ob1 =resultset =ob2))))))
        ($writeer concat performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.318   0.5

(token-11 ((result set (=rel =resultset))
        (MATCHLIST (=ob1 present unmarked) (=ob2 present unmarked)))
        -->
        (($delete-from wm
                (MATCHLIST (=ob1 present unmarked)
                        (=ob2 present unmarked)))
        ($delete-from wm (result set (=rel =resultset)))
        ($add-to wm
            ($newadd
             (result set
                 (=rel
                 (($perform delete_insert
                         =ob1
                         =resultset
                         =ob2))))))
        ($writeer delete_insert performed)
        ($add-to wm ($newadd (FLAG (result-set-modified))))))
0.348   0.6
```

Table 1

Operators used in the model

Name Args	Preconditions	Effect
delete (ob; list)	ob is a member of the list.	Remove one occurence of ob from list.
join(list1; list2)	Neither list is empty.	Combine ob1 and ob2 into a single list.
get_front(ob; list)	ob is an element of the list.	Return the part of list preceding ob.
get_rest(ob; list)	ob is an element of the list.	Return the part of the list following ob.
append(ob1; ob2; list)	ob1 in list; ob2 not in list.	Make ob2 the immediate successor of ob1.
insert(ob1; ob2; list)	ob1 not in list; ob2 in list.	Make ob1 the immediate predecessor of ob2 in the list.
delete_append(ob1; ob2; list)	ob1 and ob2 both in list.	Make ob1 the immediate successor of ob2.
delete_insert(ob1; ob2; list)	ob1 and ob2 both in list.	Make ob1 the immediate predecessor of ob2.
switch(ob1; ob2; list)	ob1 and ob2 both indeterminate.	Exchange position of ob1 and ob2 and place markers around
mark_indet(ob1; ob2; list)	no markers currently in list.	the indeterminate parts of a list.
remove_indet_markers(list)	-	Remove indeterminacy markers from a list.
append_indet(ob1; ob2; list)	ob2 not in list; ob1 in list.	Make ob2 the immediate successor of ob1 and mark ob2 as indeterminate with respect to the tail of the list.
insert_indet(ob1; ob2; list)	ob1 not in list; ob2 in list.	Make ob1 the immediate predecessor of ob2 and mark ob1 as indeterminate with respect to the preceding portion of the list.
concat_indet(ob1; ob2; list)	Neither object already in list.	Create new list segment (ob1 ob2) marking it as indeterminate with respect to existing list.

Table 2

Hypothesised processes for 3-term and 4-term problems

		Problem Form	Solution with integration	Most common solution without integration
Three-term adjacent-and-nonadjacent	1.	AB, BC, AC	ABC	ACB
	2.	BC, AB, AC	ABC	ACB
	3.	AB, AC, BC	ABC	ABC
	4.	BC, AC, AB	ABC	ABC
	5.	AC, AB, BC	ABC	ACB
	6.	AC, BC, AB	ABC	ACB
Four-term adjacent only	7.	AB, BC, CD	ABCD	ABCD
	8.	AB, CD, BC	ABCD	ABCD
	9.	BC, AB, CD	ABCD	ABCD
	10.	BC, CD, AB	ABCD	ABCD
	11.	CD, AB, BC	ABCD	BCDA
	12.	CD, BC, AB	ABCD	ABCD

Table 3

Four sets of twelve problem forms.

Premise	Resultset	Effort	Production fired
a > b	a b	0.6	
b > c	a b c	0.6	token-1
a > c	a c b	0.6	token-2
b > c	b c	0.95	
a > b	a b c	0.95	token-3
a > c	a c b	1.213	token-2
a > b	a b	1.409	
a > c	a << c ^ b >>	1.409	token-12
b > c	a b c	1.409	token-13
b > c	b c	1.402	
a > c	<< b ^ a >> c	1.402	token-14
a > b	a b c	1.402	token-15
a > c	a c	1.395	
a > b	a << b ^ c >>	1.395	token-12
b > c	a b c	1.395	token-16
a > c	a c	1.388	
b > c	<< a ^ b >> c	1.388	token-14
a > b	a b c	1.388	token-17
a > b	a b	1.381	
b > c	a b c	1.381	token-1
c > d	a b c d	1.381	token-1
a > b	a b	1.374	
c > d	<< a b ^ c d >>	1.374	token-18
b > c	a b c d	1.374	token-19
b > c	b c	1.368	
a > b	a b c	1.368	token-3
c > d	a b c d	1.368	token-1
b > c	b c	1.361	
c > d	b c d	1.361	token-1
a > b	a b c d	1.361	token-3
c > d	c d	1.354	
a > b	<< c d ^ a b >>	1.354	token-18
b > c	a b c d	1.354	token-13
c > d	c d	1.347	
b > c	b c d	1.347	token-3
a > b	a b c d	1.347	token-3
a > b	a b	1.34	
b > c	a b c	1.34	token-1
a > c	a b c	1.34	token-20
b > c	b c	1.334	

a > b	a b c	1.334	token-3
a > c	a b c	1.334	token-20
a > b	a b	1.327	
a > c	a << c ^ b >>	1.327	token-12
b > c	a b c	1.327	token-13
b > c	b c	1.32	
a > c	<< b ^ a >> c	1.32	token-14
a > b	a b c	1.32	token-15
a > c	a c	1.314	
a > b	a << b ^ c >>	1.314	token-12
b > c	a b c	1.314	token-16
a > c	a c	1.307	
b > c	<< a ^ b >> c	1.307	token-14
a > b	a b c	1.307	token-17
a > b	a b	1.301	
b > c	a b c	1.301	token-1
c > d	a b c d	1.301	token-1
a > b	a b	1.294	
c > d	<< a b ^ c d >>	1.294	token-18
b > c	a b c d	1.294	token-19
b > c	b c	1.288	
a > b	a b c	1.288	token-3
c > d	a b c d	1.288	token-1
b > c	b c	1.281	
c > d	b c d	1.281	token-1
a > b	a b c d	1.281	token-3
c > d	c d	1.275	
a > b	<< c d ^ a b >>	1.275	token-18
b > c	a b c d	1.275	token-13
c > d	c d	1.269	
b > c	b c d	1.269	token-3
a > b	a b c d	1.269	token-3
a > b	a b	1.262	
b > c	a b c	1.262	token-1
a > c	a b c	1.262	token-20
b > c	b c	1.256	
a > b	a b c	1.256	token-3
a > c	a b c	1.256	token-20
a > b	a b	1.25	
a > c	a << c ^ b >>	1.25	token-12
b > c	a b c	1.25	token-13
b > c	b c	1.243	
a > c	<< b ^ a >> c	1.243	token-14
a > b	a b c	1.243	token-15

a > c	a c	1.237	
a > b	a << b ^ c >>	1.237	token-12
b > c	a b c	1.237	token-16
a > c	a c	1.231	
b > c	<< a ^ b >> c	1.231	token-14
a > b	a b c	1.231	token-17
a > b	a b	1.225	
b > c	a b c	1.225	token-1
c > d	a b c d	1.225	token-1
a > b	a b	1.219	
c > d	<< a b ^ c d >>	1.219	token-18
b > c	a b c d	1.219	token-19
b > c	b c	1.213	
a > b	a b c	1.213	token-3
c > d	a b c d	1.213	token-1
b > c	b c	1.207	
c > d	b c d	1.207	token-1
a > b	a b c d	1.207	token-3
c > d	c d	1.201	
a > b	<< c d ^ a b >>	1.201	token-18
b > c	a b c d	1.201	token-13
c > d	c d	1.195	
b > c	b c d	1.195	token-3
a > b	a b c d	1.195	token-3
a > b	a b	1.189	
b > c	a b c	1.189	token-1
a > c	a b c	1.189	token-20
b > c	b c	1.183	
a > b	a b c	1.183	token-3
a > c	a b c	1.183	token-20
a > b	a b	1.177	
a > c	a << c ^ b >>	1.177	token-12
b > c	a b c	1.177	token-13
b > c	b c	1.171	
a > c	<< b ^ a >> c	1.171	token-14
a > b	a b c	1.171	token-15
a > c	a c	1.165	
a > b	a << b ^ c >>	1.165	token-12
b > c	a b c	1.165	token-16
a > c	a c	1.159	
b > c	<< a ^ b >> c	1.159	token-14
a > b	a b c	1.159	token-17
a > b	a b	1.153	
b > c	a b c	1.153	token-1
c > d	a b c d	1.153	token-1

```
a > b      a b                  1.148
c > d      << a b ^ c d >>      1.148      token-18
b > c      a b c d              1.148      token-19

b > c      b c                  1.142
a > b      a b c                1.142      token-3
c > d      a b c d              1.142      token-1

b > c      b c                  1.136
c > d      b c d                1.136      token-1
a > b      a b c d              1.136      token-3

c > d      c d                  1.13
a > b      << c d ^ a b >>      1.13       token-18
b > c      a b c d              1.13       token-13

c > d      c d                  1.125
b > c      b c d                1.125      token-3
a > b      a b c d              1.125      token-3
```

Table 4

Indeterminate Problem forms used in Experiment 3

Three-term indeterminates

	13	14
	A B A C A B	A C B C A C
possible orders	A B C A C B	A B C B A C

Four-term indeterminates

	15	16	17	18
	B C C D A C	B C A B B D	A B A C B D	B D C D A B
possible orders	A B C D B A C D	A B C D A B D C	A B C D A C B D A B D C	C A B D A C B D A B C D

Table 5

Commonly Used Strategies, Showing Patterns of Correctness Judgements in Response to Suggested Orders

Premises	Computer Orders			Strategies													
				Integrate		Front		Back		Adjacent		Remote		Adj/con		Sequential	
				2[*]	3[**]	2[*]	3[**]	2[*]	3[**]	2[*]	3[**]	2[*]	3[**]	2[*]	3[**]	2[*]	3[**]
AB, BC, AC	ab	abc	acb[x]	a	r	i	a	i	r	a	a	a	r	a	r	a	r
	ab	abc	abc	a	a	i	a	i	a	a	r	a	a	a	a	a	a
	ab	abc	bac[x]	a	r	i	r	i	a	a	a	a	r	a	r	a	r
BC, AB, AC	bc	abc	abc	a	a	i	a	i	a	a	a	a	a	a	a	a	a
	bc	abc	acb[x]	a	r	i	a	i	r	a	r	a	r	a	r	a	r
	bc	abc	bac[x]	a	r	i	r	i	a	a	r	a	r	a	r	a	r
AB, AC, BC	ab	abc	abc	i	a	a	a	i	a	r	a	a	r	r	a	a	a
	ab	acb	bac[x]	i	r	a	r	i	a	a	r	r	a	a	r	r	r
AC, AB, BC	ac	abc	bac[x]	i	r	a	r	i	a	a	r	r	a	a	r	r	r
	ac	acb	abc	i	a	a	a	i	a	r	a	a	r	r	a	a	a
BC, AC, AB	bc	abc	acb[x]	i	r	i	a	a	r	r	r	a	a	r	r	r	r
	bc	bac	abc	i	a	i	a	a	a	a	a	r	r	a	a	a	a
AC, BC, AB	ac	bac	acb[x]	i	r	i	a	a	r	r	r	a	a	r	r	r	r
	ac	abc	abc	i	a	i	a	a	a	a	a	r	r	a	a	r	r
CD, AB, BC	cd	cdab	bcda[x]	i	r	i	i	i	r	a	a	r	a	a	r	a	r
	cd	cdab	abcd	i	a	i	i	i	i	a	a	r	a	a	r	a	a[***]

Note. The judgements are accepting (a) the suggested order as correct, rejecting (r) it as incorrect or indicating that the solution is indeterminate (i).

[*] Judgement made after the second premise is presented.

[**] Judgement made after the third premise.

[***] Same as integration, except that the indeterminacies are resolved in the order the terms appear.

[x] Denotes that the computer gives the wrong order.

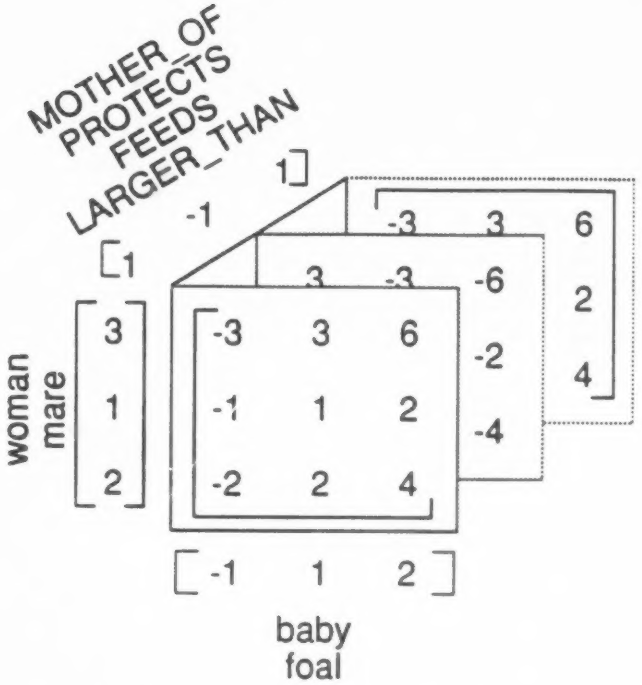
MOTHER_OF
PROTECTS
FEEDS
LARGER_THAN
[1 -1 1]
woman
mare
[3 1 2]
-3 3 6
-1 1 2
-2 2 4
3 -3 -6
-2
-4
-3 3 6
2
4
[-1 1 2]
baby
foal

Figure 1. Tensor product representation of predicate-argument binding.

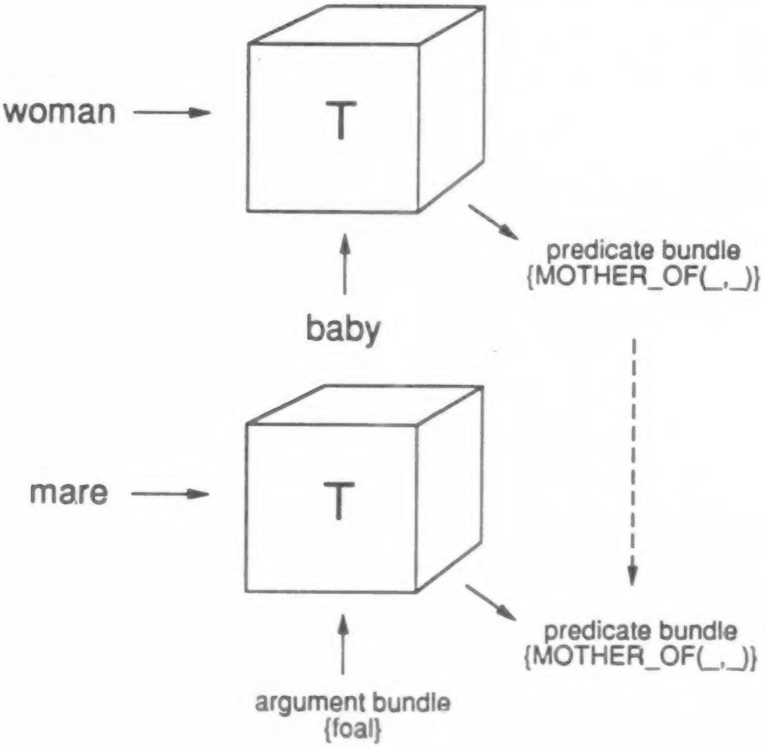
woman
T
baby
predicate bundle
{MOTHER_OF(_,_)}
mare
T
argument bundle
{foal}
predicate bundle
{MOTHER_OF(_,_)}

Figure 2. Solution process for simple analogy in STAR model.

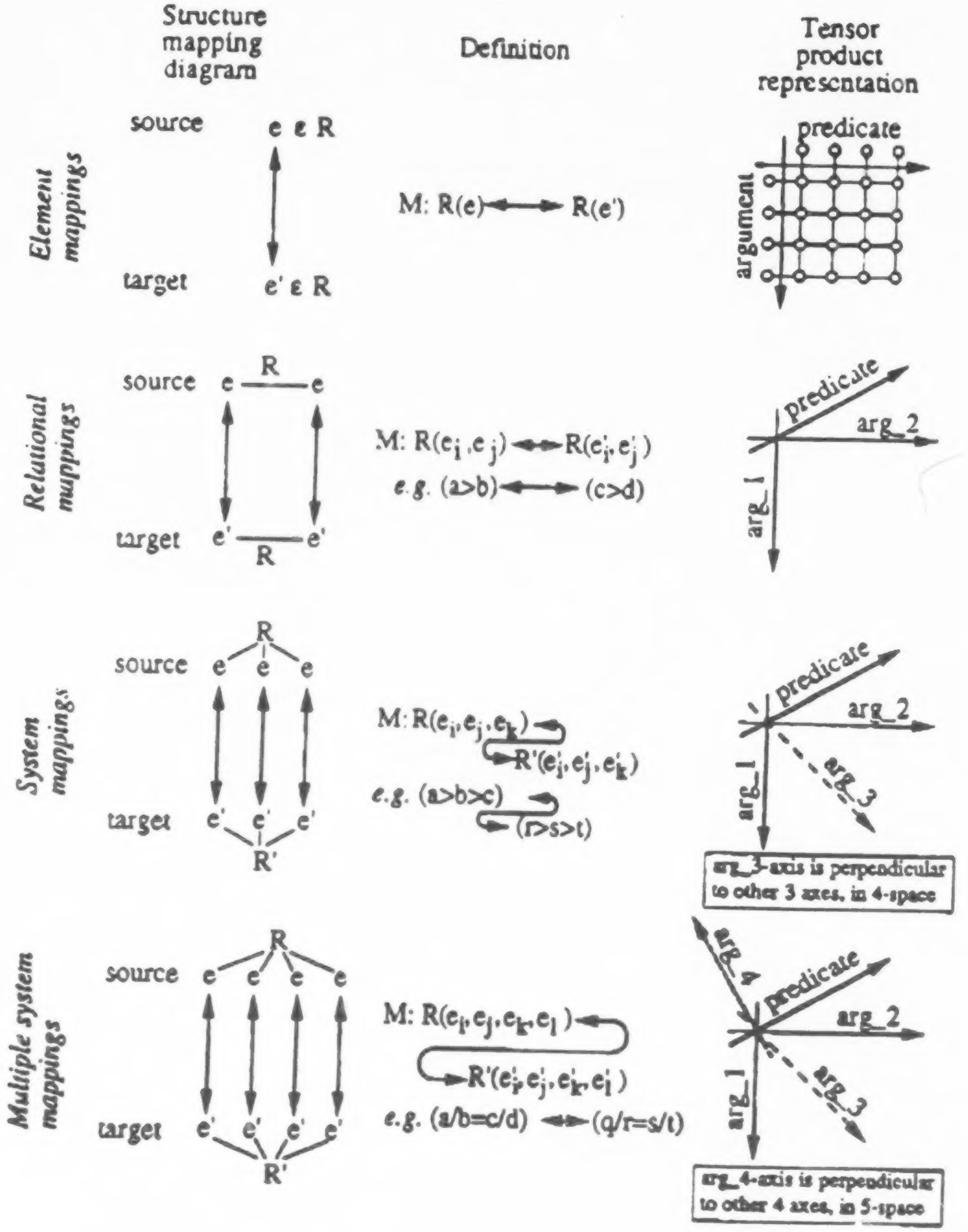
Structure mapping diagram
Definition
Tensor product representation
Element mappings
source
e ε R
target
e' ε R
M: R(e) ⟷ R(e')
predicate
argument
Relational mappings
source
e — R — e
target
e' — R — e'
M: R(e_i, e_j) ⟷ R(e'_i, e'_j)
e.g. (a>b) ⟷ (c>d)
predicate
arg_2
arg_1
System mappings
source
R
e e e
target
e' e' e'
R'
M: R(e_i, e_j, e_k) ⟷ R'(e'_i, e'_j, e'_k)
e.g. (a>b>c) ⟷ (r>s>t)
predicate
arg_2
arg_1
arg_3
arg_3-axis is perpendicular to other 3 axes, in 4-space
Multiple system mappings
source
R
e e e e
target
e' e' e' e'
R'
M: R(e_i, e_j, e_k, e_l) ⟷ R'(e'_i, e'_j, e'_k, e'_l)
e.g. (a/b=c/d) ⟷ (q/r=s/t)
arg_4
predicate
arg_2
arg_1
arg_3
arg_4-axis is perpendicular to other 4 axes, in 5-space

Figure 3. Dimensionality of representation required for levels of structure mapping

BEST COPY AVAILABLE

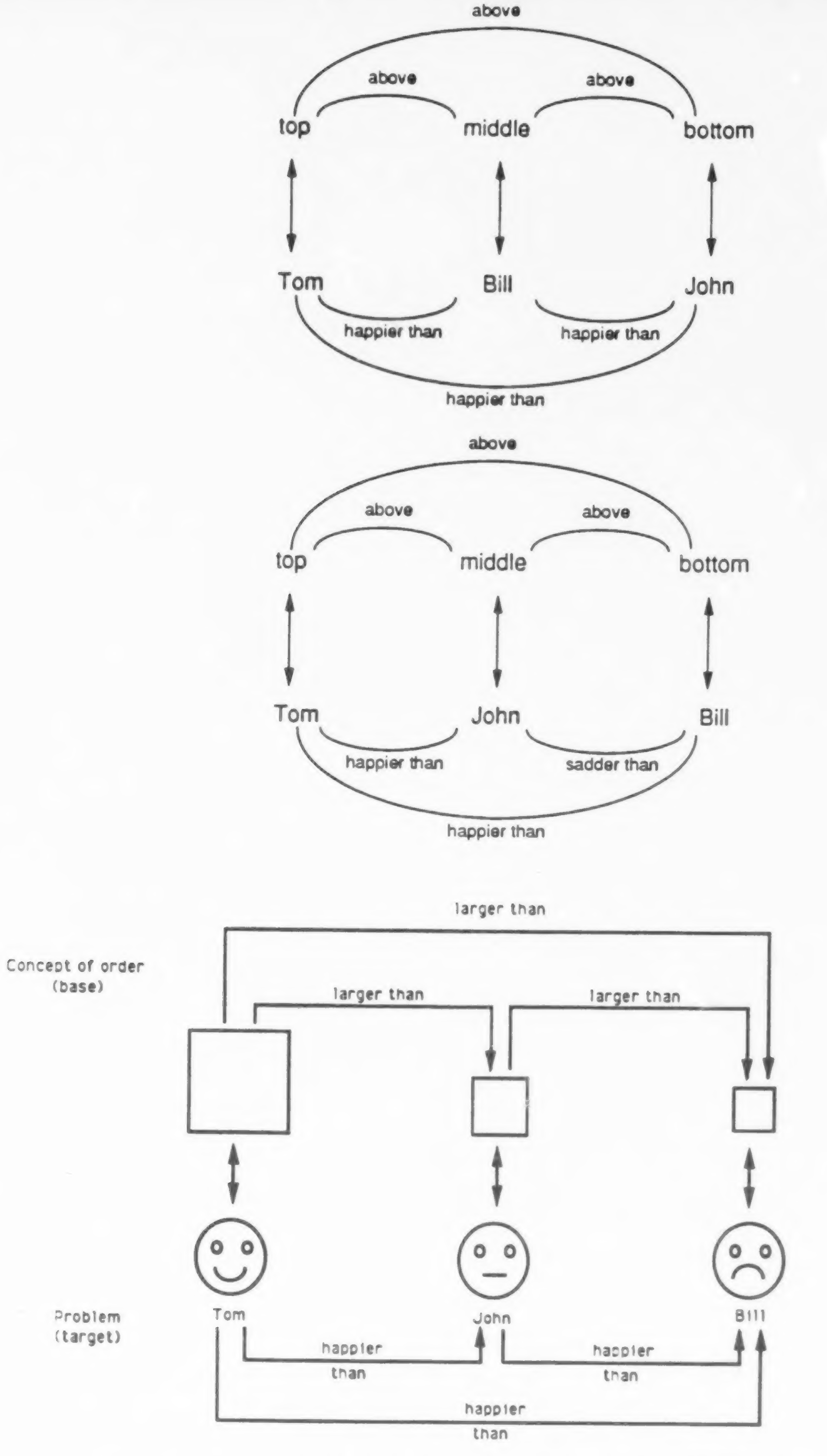
above
above
above
top
middle
bottom
Tom
Bill
John
happier than
happier than
happier than
above
above
above
top
middle
bottom
Tom
John
Bill
happier than
sadder than
happier than
larger than
Concept of order
(base)
larger than
larger than
Problem
(target)
Tom
John
Bill
happier
than
happier
than
happier
than

Figure 4. Consistent (A) and inconsistent (B) mapping of transitive inference problem into top-down schema, or (C) into concrete representation.

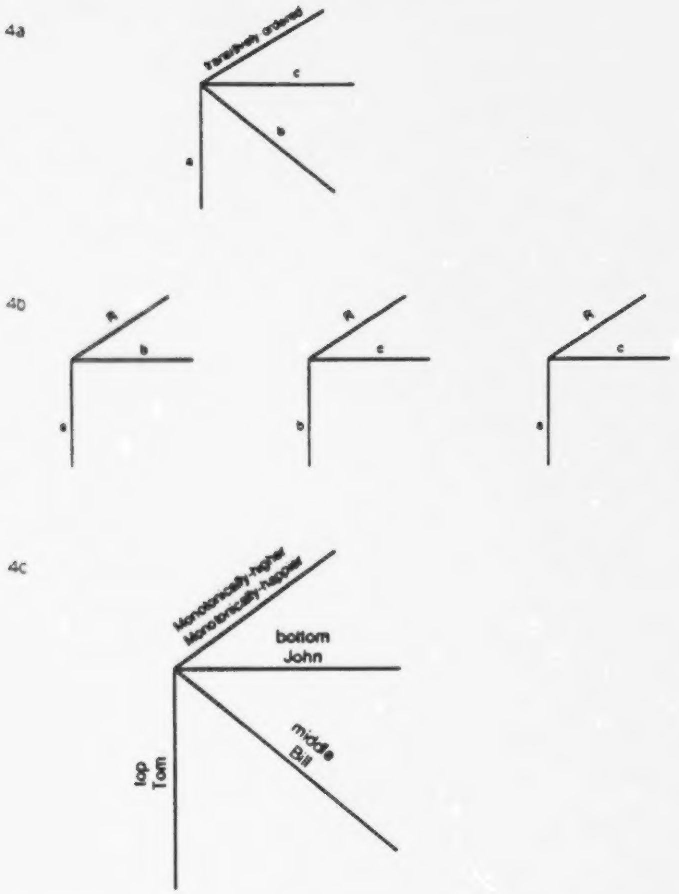
4a
transitively ordered
c
b
a
4b
R
b
a
R
c
b
R
c
a
4c
Monotonically-higher
Monotonically-happier
bottom
John
middle
Bill
top
Tom

Figure 5. Tensor product representation of transitive inference.

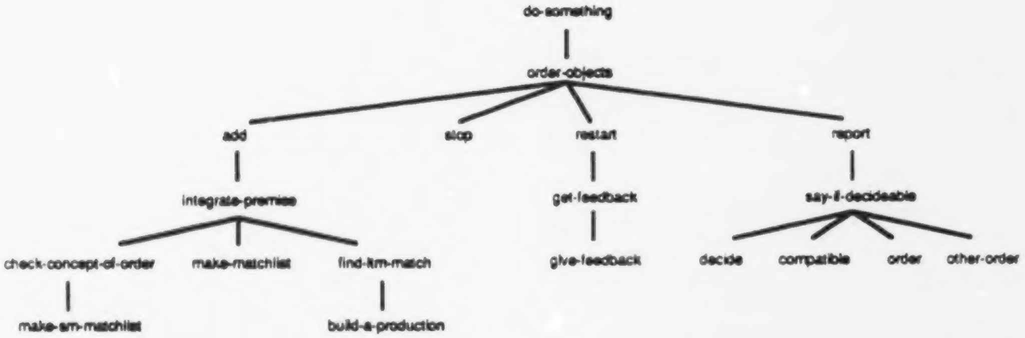
do-something
order-objects
add
stop
restart
report
integrate-premise
get-feedback
say-if-decideable
check-concept-of-order
make-matchlist
find-ltm-match
give-feedback
decide
compatible
order
other-order
make-stm-matchlist
build-a-production

Figure 6. Goal hierarchy.

Constructed Order	Premise	Action
- - -	a > b	insert in WM
a b	b > c	append
a b c		

Figure 7. Sequence of steps in simple transitive inference problem.

Production built:

(((constructed order (<relation> <obj 1. . . . . . obj n>))

(new premise <obj 1> <relation> <obj 2>)

(match obj 1 obj n))

$\longrightarrow$

(add-to WM (perform APPEND <obj n> <obj 2>)))

Instantiation:

working-memory contents	premise	new working-memory contents
ab	b>c	abc

Figure 8. Production built in performing problem a>b, b>c.

	constructed order	premise	action
Relational Strategy	- - -	c > d	insert in WM
	c d	a > b	concatenate
	c d a b	b >c	delete - insert
	b c d a		
System Strategy	- - -	c d	insert in WM
	c d	a > b	concat indet
	(cd)^(ab)	b > c	switch
	a b c d		

* indeterminacy marker

Figure 9. Steps in relational and system strategies.

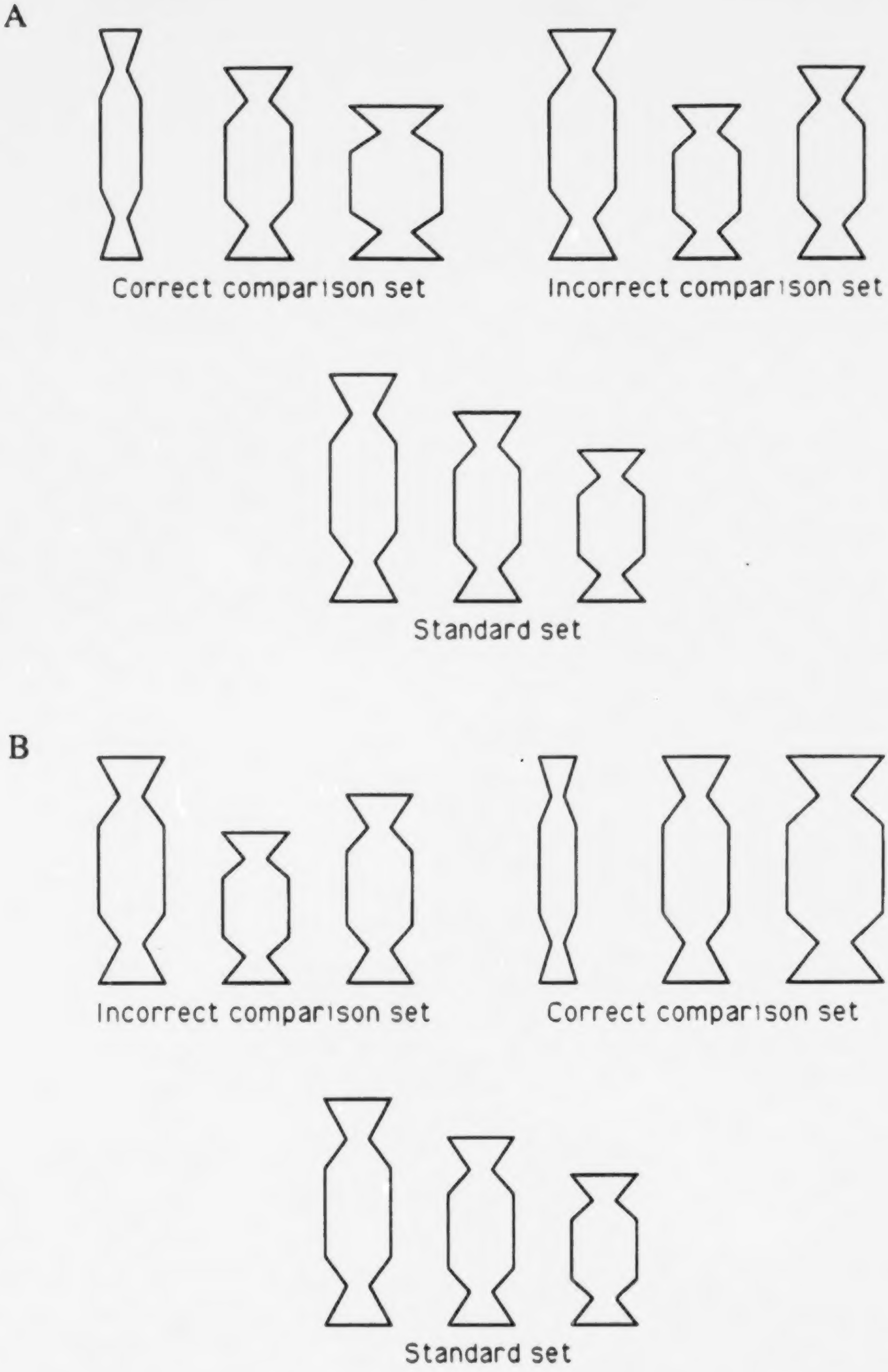
A
Correct comparison set
Incorrect comparison set
Standard set
B
Incorrect comparison set
Correct comparison set
Standard set

Figure 10. Training (A) and transfer (B) problems for ordered set mapping experiment.

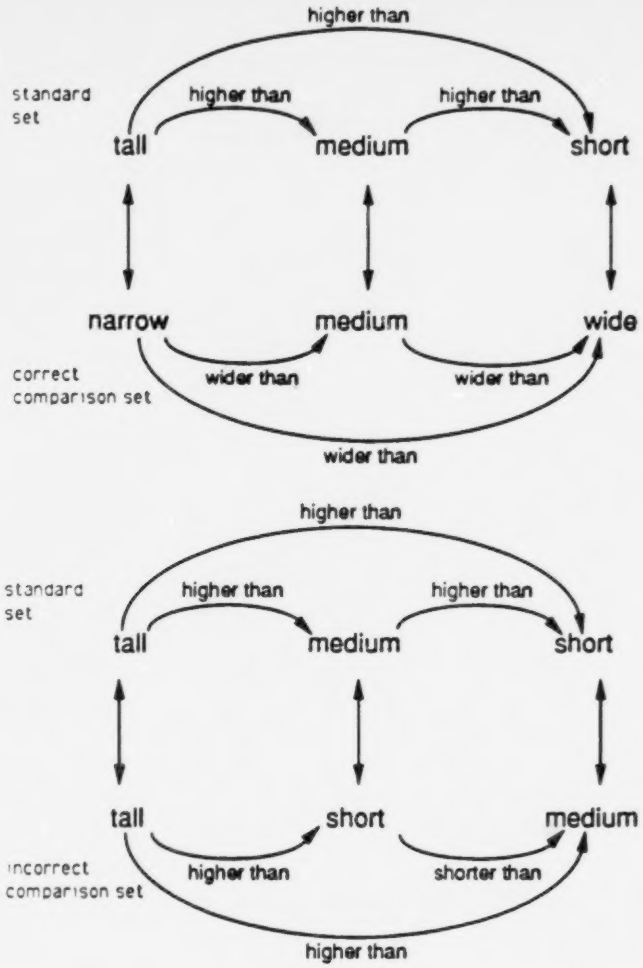
higher than
standard set
higher than
higher than
tall
medium
short
narrow
medium
wide
correct comparison set
wider than
wider than
wider than
higher than
standard set
higher than
higher than
tall
medium
short
tall
short
medium
incorrect comparison set
higher than
shorter than
higher than

Figure 11. Structure mapping required for ordered set mapping experiment.

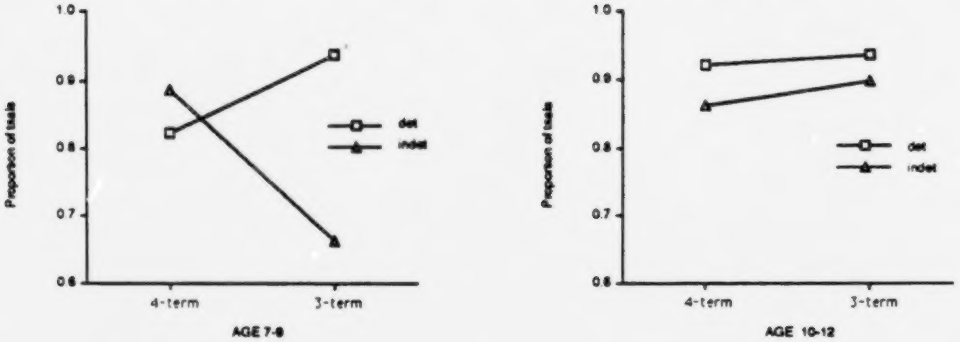

Figure 12. Proportion of correct determinacy judgments in first or second question in Experiment 3.